Mini projeto

Compiladores 2018.2

Marco Polo

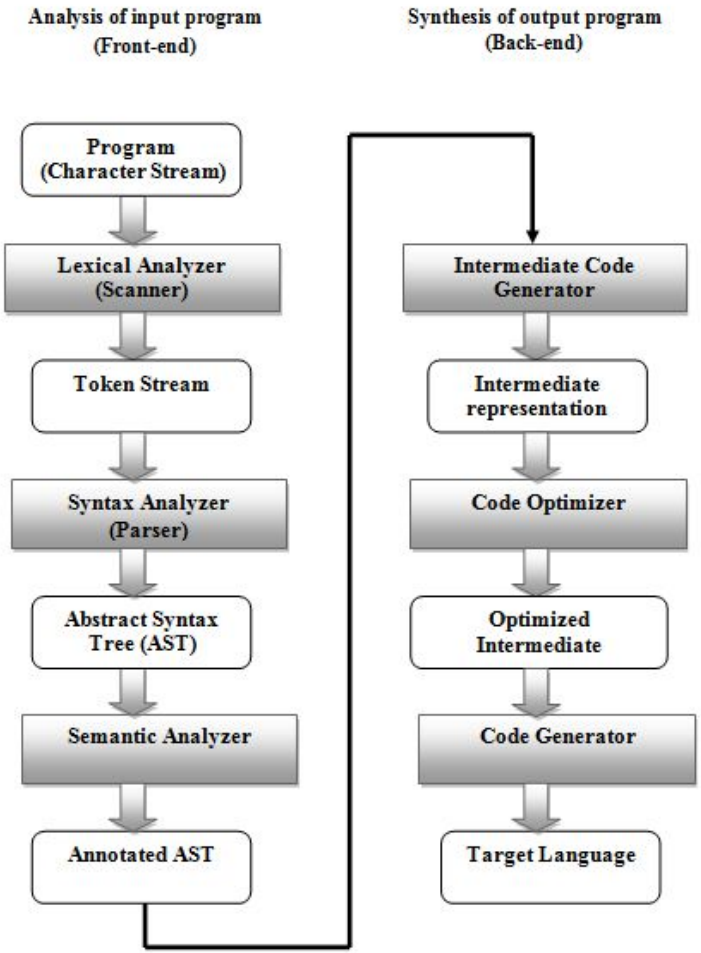
Histórico das aulas práticas

AP1
Lexer + Parser

AP2
Avaliação de expressão

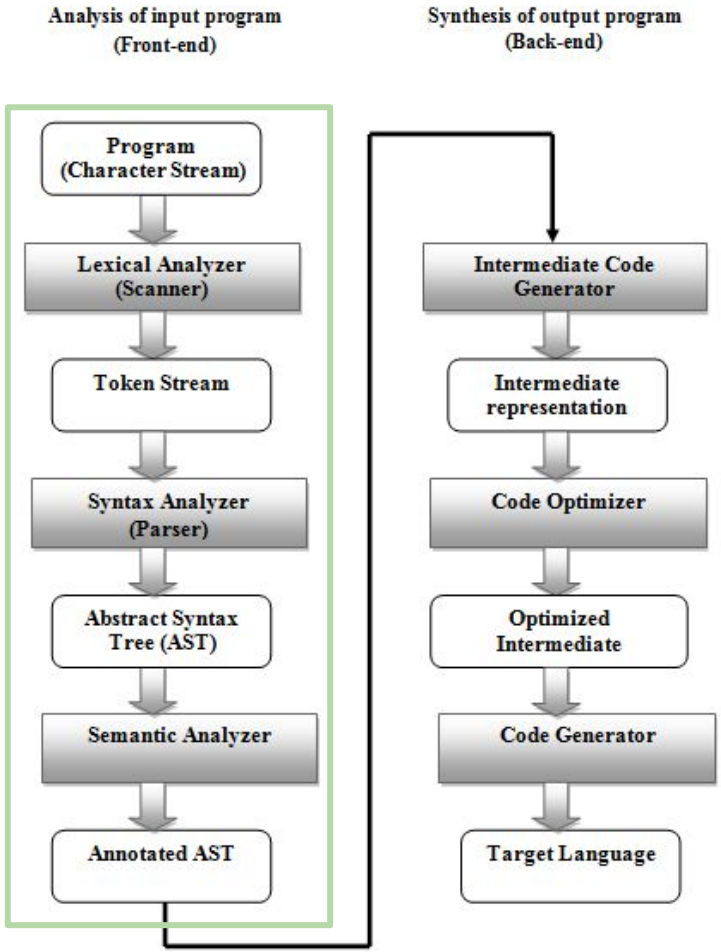
AP3
Análise semântica

Um compilador



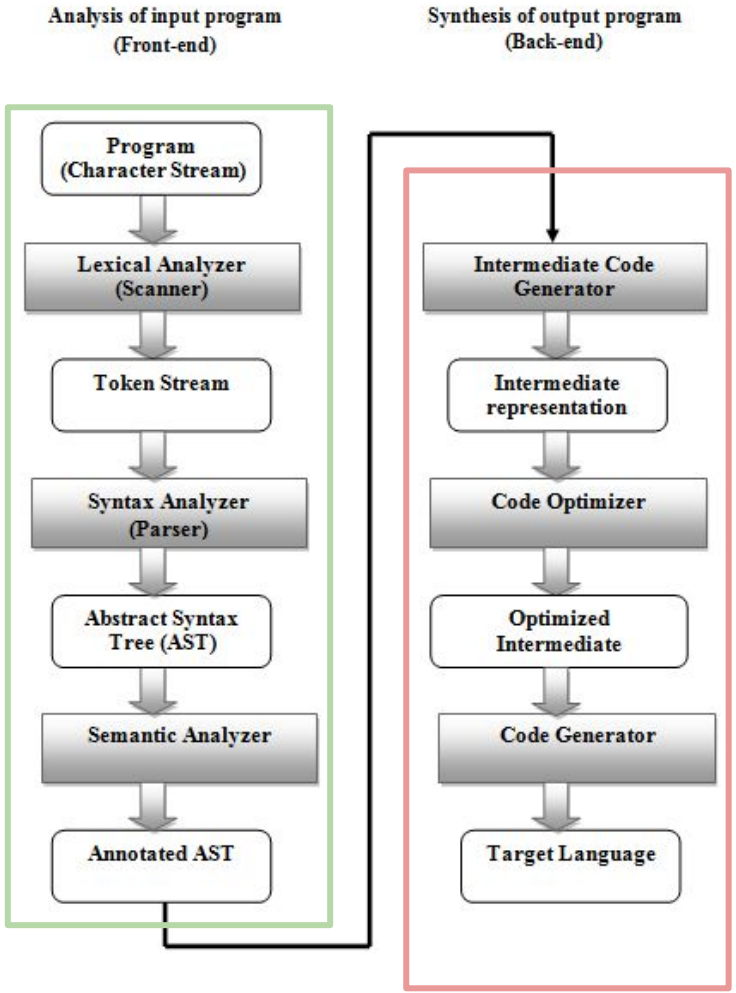
Um compilador

O que vocês já fizeram:



Um compilador

O que vocês já fizeram:



O que faltaria fazer para construir um compilador completo.

LLVM

http://llvm.org/

LLVM IR

```
int main() {
    return 42;
}
```

```
$ clang -S -emit-llvm -O3 simple.c
```

```
define i32 @main() #0 {
    ret i32 42
}
```

LLVM IR

```
int main() {
    int a = 32;
    int b = 16;
    return a + b;
}
```

LLVM IR

```
int main() {
    int a = 32;
    int b = 16;
    return a + b;
}
```

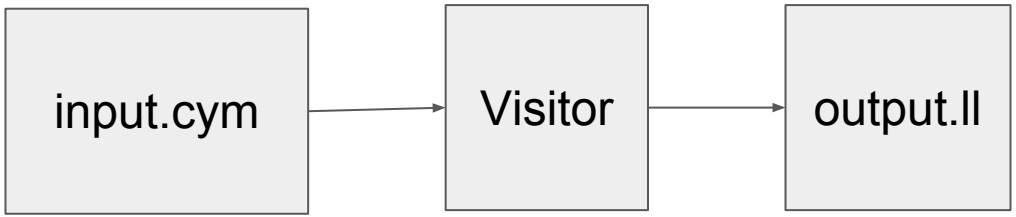
```
define i32 @main() #0 {
    %1 = alloca i32, align 4
    %a = alloca i32, align 4
    %b = alloca i32, align 4
    store i32 0, i32* %1
    store i32 32, i32* %a, align 4
    store i32 16, i32* %b, align 4
    %2 = load i32, i32* %a, align 4
    %3 = load i32, i32* %b, align 4
    %4 = add nsw i32 %2, %3
    ret i32 %4
}
```

O Projeto

ANTLR + 

O projeto

- Fazer um visitor para ler um programa escrito na linguagem Cymbol (.cym) e produzir LLVM IR (.ll) que seja compilável e executável através do compilador de LLVM IR
  - Compilador de LLVM IR: llc <input>.ll
  - Interpretador de LLVM IR: lli <input>.ll
  - Referência do LLVM IR: http://llvm.org/docs/LangRef.html
  - Emitir LLVM IR a partir de C: clang -S -emit-llvm <input>.c
    - Caso queira olhar como o Clang traduz C para LLVM IR
    - Uma boa maneira de “”tirar uma dúvida rápida” de como produzir LLVM IR



O Projeto

- O projeto pode ser feito individualmente ou em dupla
- A linguagem de implementação é de sua escolha, contanto que Antlr dê suporte a ela
  - Java, C#, Python (2 and 3), JavaScript, Go, C++, Swift
- A avaliação será feita através da apresentação do código implementado e testes de programas escritos em Cymbol.
  - Os programas serão corretos sintaticamente e semanticamente.