Relatório da fase 5:

Tomás Carreira nº50760

Para a fase 5 do projecto decidi adicionar structs à linguagem plush. A sintaxe concreta foi inspirada pelas linguagens C e Rust.

```
Struct A {
      var a: int,
      var b: float,
}

var sa: struct := A(1, 2.0);

var b: float := sa.b;

sa.a := 3;
```

Para implementar structs foi necessário fazer alterações em todas as fases do compilador.
No *lexer* foi necessário adicionar os tokens ‘struct’ e ‘.’. No *parser* foi necessário adicionar regras para aceitar declarar struct, inicializar structs, ler campos de structs e escrever em campos de structs (os quatro exemplos acima). Foi necessário, também, mudar a regra dos nomes dos tipos para aceitar ‘struct *nome_da_struct*’. Na primeira passagem do *type checker* foi adicionado ao contexto as declarações de structs. Na segunda passagem é verificado a inicialização da struct, se o número de campos e o tipo dos campos está correcto, também é verificado que os acessos a campos de struct são feitos a structs, os campos existem e que têm o tipo correcto. Durante o type checking é adicionado à árvore os tipos resultantes de aceder a um campo e qual o seu index na struct para facilitar a geração de código. Com a informação recolhida no *type checking* a geração de código não foi muito difícil. Para a declaração da struct foi preciso adicionar uma declaração com os tipos que compõe a struct. Para inicializar a struct foi usado um *alloca* para alocar espaço em memória e foi usado um getelementptr para popular a struct com os valores. Para aceder aos campos é usado getelementptr e dependendo se é leitura ou escrita é usado um load ou store.

Uma grande dificuldade que tive foi implementar structs dentro de structs que foi superada com muita paciência e um código python muito feio.

Um problema com esta implementação das structs é o ffi. Se for usada uma função externa em que os argumentos ou o return são structs não há garantias de que a organização dos dados das structs é igual em c e em plush. Isto podia ser resolvido usando a funcionalidade em llvm chamada *data layout* que permite especificar com é que os dados são guardados em memória. Para além disso seria necessário identificar os tipos externo e declará-los como tipos opacos.