# Trabalho prático 2 - Compiladores

## Documentação - Ligador

**Luiz Felipe Gontijo, Marcos Vinicius, Matheus Pimenta**

### 1. Introdução

Este trabalho tem como objetivo fixar os conceitos da disciplina de Compiladores, especialmente sobre ligação de programas. A atividade proposta foi construir um editor de ligação para uma máquina virtual previamente disponibilizada, onde dado um ou mais programas com instruções em Assembly, deve-se montá-los separadamente (com o montador do TP1 adaptado) e ligar os vários módulos gerados em um único programa objeto, realocando-os na memória.

### 2. Implementação

Antes de começar a desenvolver o ligador, foi necessário fazer adaptações no montador. Diferente do TP1, o montador deve gerar um arquivo intermediário, com uma tabela de símbolos do programa recebido na entrada, para que o ligador consiga juntar os programas. As alterações no montador estão descritas abaixo:

### 2.1 Montador

No final da segunda fase do montador, após feita a tradução de tudo o que foi possível e antes de imprimir as instruções traduzidas na saída, o montador passa por toda a tabela dos símbolos encontrados na fase 1 duas vezes. Na primeira, são contados quantos símbolos foram identificados, imprimindo essa quantidade na primeira linha do arquivo de saída. Já na segunda, cada símbolo identificado é imprimido da seguinte forma:

*<label>:<posição no programa>*

Por exemplo, caso esteja definido uma constante “x: WORD 2” na posição 10 do programa, na saída do montador terá uma linha com “x:10”.

Os símbolos que não foram encontrados no final não são traduzidos no programa, e portanto são imprimidos do jeito que foram encontrados. É tarefa do ligador encontrar onde está definido esse símbolo e substituir por um apontamento, após juntar os programas.

## 2.2 Ligador

A implementação do ligador seguiu algumas diretrizes em que foram definidas estruturas e classes que seguem o comportamento do front-end de um compilador. Assim, criamos um *Lexer* para melhor abstrair e modularizar as funções do ligador, e uma estrutura *File* para armazenar informações dos arquivos.

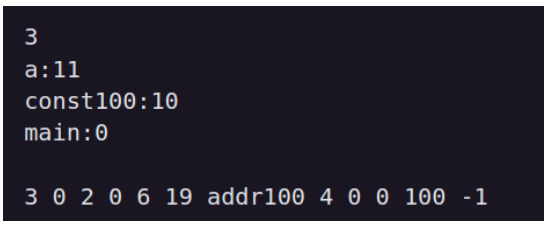
Para cada arquivo recebido pelo ligador, o *Lexer* lê a primeira linha, que contém o número de símbolos definidos naquele arquivo. Em seguida, são lidos todos os símbolos com suas posições, e armazenados numa tabela. Além das posições relativas dos símbolos, também é guardado na tabela o arquivo a qual o símbolo pertence. Por fim, é armazenado num vetor o programa semi-traduzido.

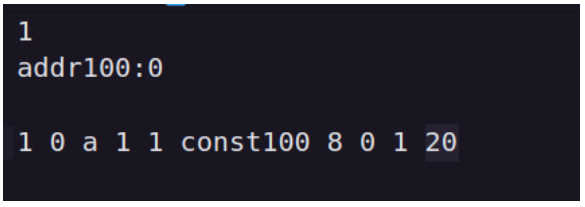
Ao final da leitura de todos os arquivos, teremos todos os programas juntos num vetor, e todos os símbolos definidos em todos os programas numa tabela. A classe portanto passa por todo o vetor com os programas, atualizando os símbolos que eram desconhecidos e atualizando as posições de memória. Por exemplo, se no programa isolado o símbolo estava na posição 0, na junção dos programas essa posição tem que ser diferente, caso esse programa não seja o primeiro que o ligador recebeu. Por fim, é impresso o "MV-EXE", o header e o programa traduzido com os endereçamentos devidamente atualizados.

## 3. Testes

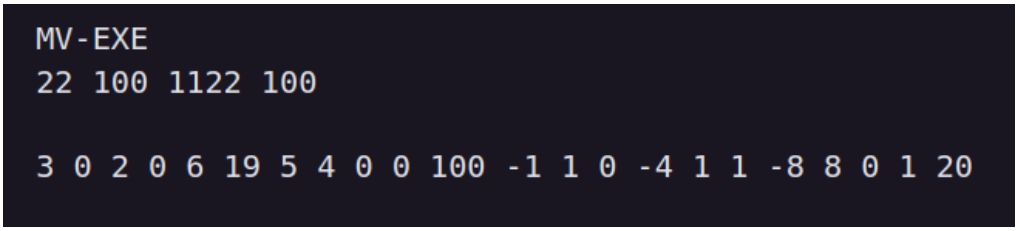
Foram executados vários casos de teste. Dentre eles, casos possuindo múltiplas linhas com comentários, código com mais de um breakline entre as linhas e quantidade variável de espaços entre as palavras, e passando os programas em ordens diferentes para o ligador.

Foi implementado na linguagem Assembly da máquina virtual alguns testes, à exemplo do testes que somam valores à entrada do programa, que se encontra na pasta "tst". Abaixo temos um exemplo de como deve ser a entrada do ligador, e a saída após a ligação:




**Figuras 1 e 2. Formato do arquivo de entrada do ligador**




**Figuras 3. Saída do ligador para os programas das figuras 1 e 2**

## 4. Conclusão

Neste trabalho foi possível implementar um ligador, assim como compreender o funcionamento desse componente do compilador, de forma a aplicar os conceitos aprendidos nas aulas. Também, praticamos conceitos que envolvem o desenvolvimento de estruturas de dados para ler e dar significado aos símbolos de uma linguagem de programação vindos de diferentes fontes.