E-fólio B | Folha de resolução para E-fólio

UNIDADE CURRICULAR: Compilação

CÓDIGO: 21018

DOCENTE: Professor Jorge Morais

A preencher pelo estudante

NOMES e N.º DE ESTUDANTE:

- Gonçalo Caraça, 2000130 (Alt + C ++ Elite)
- Inês Oliveira, 2001090 (Alt + C ++ Elite)
- Joana Martins, 2003351 (Alt + C ++ Elite)
- João Carvalho, 2103537 (Alt + C ++ Elite)
- Mário Carvalho, 2000563 (Alt + C ++ Elite)

CURSO: Licenciatura em Engenharia Informática

DATA DE ENTREGA: 01 de junho de 2023

TRABALHO / RESOLUÇÃO:

INTRODUÇÃO

De acordo com o enunciado do e-folio B, compreender a forma como o código intermediário é gerado e otimizado, foi por vezes uma dificuldade que exigiu dedicação e tempo. Para isso recorreu-se a simplificação do código, ordenação de funções e simplificação de variáveis, usando as ferramentas e linguagens apropriadas, para a implementação das fases da geração de código intermédio.

Após a entrega do e-fólio A, começamos a implementação da árvore TAC e transformação para assembly, e posteriormente fizemos várias correções de erros verificados na avaliação anterior.

DESENVOLVIMENTO

Após a análise léxica e sintática do compilador, a equipa propôs-se à correção dos "*warnings*" do código implementado, assim como à remoção de código redundante. Foi também efetuada a correção da problemática dos operadores aritméticos invertidos (Exemplo: "+=" em vez de "=+"). Adicionalmente, também corrigimos o problema de não interpretar o sinal negativo nos números inteiros ou decimais.

Além disso, foi efetuada a restrição do número de *structs*, *const* e globais para 1, assim como *locals* a 1 dentro de cada função, o que levou à reestruturação de todo o código já desenvolvido tanto no analisador sintático como lexical.

Nesta segunda fase de otimização e geração de código intermédio, para armazenar e manipular informações recorremos a uma pilha. Ao usar esta combinação o *Flex* permite identificar os *tokens*, convertendo-os numa sequência de símbolos reconhecidos. Em seguida, o *Bison* utiliza esses símbolos para construir uma árvore sintática, seguindo as regras de gramática definidas. Recorreu-se à implementação de uma pilha (FIFO) que ajudou a

armazenar dados temporariamente e a recuperá-los conforme necessário, para serem utilizados posteriormente.

DIFICULDADES

Face ao enunciado, ainda ficaram alguns pontos neste e-fólio por resolver, nomeadamente a verificação do tamanho dos vetores.

Após a reestruturação do código face aos pontos identificados na avaliação, tivemos dificuldades em colocar a implementação TAC a funcionar novamente, pelo que mantemos a transformação assembly criada na primeira versão, e a segunda tentativa para a implementação TAC.

O exemplo a seguir apresenta um teste simples da implementação. O nosso objetivo seria dar continuidade à conversão do código num ficheiro intermédio, para depois permitir a otimização em assembly, o que iria gerar dois ficheiros para o código intermédio e o código em assembly.

```
$ ./teste t
const encontrado
variavel a do tipo int com valor -20
variavel b do tipo int com valor 2
$t1 = -20
a = $t1
$t2 = 2
b = $t2

Programa sem erros.
```

Figura 1 - Teste simples de implementação TAC

Além disso, ainda não conseguimos avaliar a funcionalidade do vetor como esperado, dado que permanece a dificuldade em relacionar o tamanho do vetor com o número de elementos atribuídos.

Associado ao relatório foram enviados os ficheiros correspondentes às melhorias do e-fólio A para B:

- sintax.y
- lex.l

e as últimas alterações conseguidas descritas anteriormente:

- sintaxOtimizado.y
- lexOtimizado.l

BIBLIOGRAFIA

Aho, A., S. Lam, M., Ravi , S., & D. Ullman, J. (2008). *Compilers: principles, techniques and tools* (2ª ed.). (A. Wesley, Ed., & D. Vieira, Trad.) Universidade Federal de Minas Gerais, Brasil: Pearson.

Reis Santos, P., & Thibault, L. (2015). *Compiladores.* FCA.