

**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:** 02 de maio de 2023

**TRABALHO / RESOLUÇÃO:**

**INTRODUÇÃO**

De acordo com o enunciado, pretende-se, em grupo, construir um compilador para a linguagem YAIL, apresentando as análises léxica e sintática correspondentes.

Numa primeira fase escolhemos realizar um compilador em C++, mas ao longo do percurso, surgiram dificuldades de funcionalidade e devido às diversas incompatibilidades, alterou-se a linguagem para C, em consenso com todos os elementos do grupo.

Juntos, na nossa opinião, C, FLEX e BISON formam um conjunto de ferramentas poderosas e flexíveis para o desenvolvimento de um compilador, pois automatizam muitas das tarefas necessárias nesse processo, além disso, podem aumentar a eficiência e a precisão, embora num primeiro contacto a relação entre ambos apresentou dificuldades e exigiu empenho. Em suma, percebeu-se que a combinação dessas tecnologias permitia que o compilador fosse criado de forma mais rápida e eficiente, sem sacrificar o desempenho ou a qualidade do código.

No entanto, apesar de acreditarmos que a nossa escolha foi a mais adequada, o uso do FLEX e BISON apresentou algumas dificuldades. Foi necessário, conhecimento sólido em teoria da linguagem e gramática formal, para a estruturação e criação de gramáticas corretas e sem ambiguidade. Entender a correta diferença entre análise léxica e sintática, além de compreender a sintaxe dessas ferramentas.

Outra dificuldade foi o processo de depuração, identificar e corrigir erros no código, nem sempre foi tarefa fácil, no entanto, com dedicação, esforço e pesquisa, conseguimos ultrapassar essas dificuldades e aproveitar os benefícios destas ferramentas na construção do nosso compilador.

**DESENVOLVIMENTO**

Depois de optarmos pela escolha do FLEX, procedemos à criação da gramática necessária para a linguagem YAIL. Através do ficheiro “*lex.l*”, foram especificados os *tokens*, como por exemplo os tipos de variáveis (*INT*, *FLOAT*, *BOOL*), os nomes dos caracteres (*ABREPARENT*, *FECHAPARENT*, etc.), ou os nomes dos métodos *default* da linguagem (*write(), write\_all()*, *write\_string()*, *etc*), por exemplo.

Além disso, neste ficheiro também estão presentes os lexemas aceites através de expressões *regex*, para que seja possível o compilador interpretar parágrafos, ou até mesmo aceitar os caracteres definidos para os nomes das variáveis (identificadores).

Através do comando “*flex lex.y*”, é criado um ficheiro em linguagem C, “*lex.yy.c*”, que representa o analisador léxico para a gramática que o nosso grupo especificou em “*lex.l*”.

No ficheiro “*syntax.y*” especificamos a gramática a ser analisada. Começamos por incluir todos os *tokens* aceites gerados pelo analisador léxico, como os tipos e características específicas da linguagem YAIL. A construção da análise léxica foi elaborada por camadas, de modo a facilitar o que o analisador pode aceitar para a linguagem nos diferentes níveis definidos. Na primeira camada é aceite o método *main(),* *structs{}*, *global*, ou comentários, por exemplo, mas nunca poderão ser aceites instruções como condições *if-else*, ou *while*, dado que estes apenas são válidos dentro de funções, como é possível visualizar no exemplo infra.

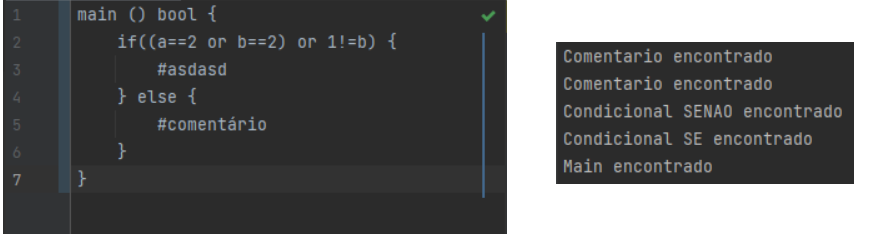


Figura - Programa YAIL sem erros

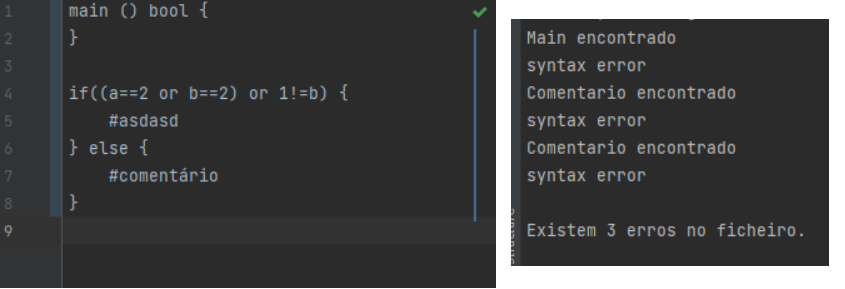


Figura - Programa YAIL com erros

Para melhor compreensão do que se realiza durante a leitura do próprio, adaptaram-se as mensagens de alerta de erros, tais como identificação da linha e o tipo de *token* onde se encontra potencialmente um erro. Indicando também a quantidade de erros detetados.

Adicionalmente, também optamos por incluir *printf’s* na consola com a gramática detetada, de modo a facilitar a visualização da interpretação do analisador.

Para compilar o analisador sintático através do analisador lexical, criamos um ficheiro *bash* “*compileEfolioA.sh*” que agrupa os comandos necessários para gerar os analisadores e fazer a análise léxica a partir um ficheiro *.txt* que contém o código da linguagem YAIL. Para os nossos testes, optamos por criar dois ficheiros, “*YAILComErros.txt*” e “*YAILSemErros.txt*” e fomos efetuando testes à medida que fizemos o desenvolvimento, com 100% de sucesso para os exemplos de “*YAILsemErros.txt*”. Foram realizados testes para todos os exemplos encontrados ao longo do enunciado, para abranger uma maior análise sobre todos os casos possíveis.

Embora esteja quase completo, avalia-se com satisfação a globalidade do resultado obtido e com grande contentamento por se ter ultrapassado o desafio de incluir funções e cálculos mais complexos tais como, potências dentro de raízes pertencentes às funções “*defaults”* de YAIL, tais como:

Uma imagem com texto

Descrição gerada automaticamente

Figura 3 - Programa YAIL sem erros para “gen”

Uma imagem com texto

Descrição gerada automaticamente

Figura 4 - Programa YAIL sem erros para “global” com métodos especiais e operações matemáticas

**MELHORIAS FUTURAS**

Como melhorias futuras pretendemos remover os “warnings” do código implementado, assim como código redundante.

Além disso, pretendemos corrigir a problemática dos operadores aritméticos invertidos (Exemplo: “+=” em vez de “=+”).

# 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.