Objectivo do Trabalho

Implemente o compilador para a linguagem *Ya!* (constituído pelos respectivos analisadores lexical, sintáctico e semântico, bem como o gerador de código MIPS, incluindo a computação do registo de activação). O compilador recebe um ficheiro `".ya"` e produz como output um ficheiro `".mips"`, caso o input não apresente erros. Caso haja erros (lexicais, sintácticos ou semânticos), o analisador deve identificá-los e mostrá-los no output. Pode usar-se a "framework" de output da APT para LaTeX previamente desenvolvida para visualizar a APT, mas não é obrigatório.

Para testar o código gerado, é sugerida a utilização do interpretador SPIM (linha de comandos) ou o MARS (com interface gráfica).

Em detalhe, devem fazer parte do compilador:

- Analisador lexical para *Ya!* (flex, jflex, sablecc, antlr, ...);
- Analisador sintáctico para *Ya!* (bison, jcup, sablecc, antlr, ...);
- Implementação da Symbol Table (ST), com "contextos" (para os "blocos" ou corpos – *coisas entre {}*);
- Analisador semântico completo, sobre a APT+ST, que devolve erros de nomes e/ou tipos.
- Gerador de código completo, incluindo as análises próprias para o registo de activação.

Descrição da linguagem *Ya!*

A linguagem *Ya!* obedece às seguintes especificações:

- Um programa é uma sequência de declarações;
- Todas as instruções são terminadas por ponto e vírgula (`;`)
- Uma declaração pode ter os seguintes formatos:
  - `i: int` (declaração de variável)
  - `i: int = 1` (declaração com valor de inicialização)
  - `i,j,k: int = 1` (declaração múltipla com valor de inicialização - todas as variáveis ficam com o mesmo valor)
  - `f(): int { <corpo> }` (declaração de função)
  - `f(a: int, b: bool): int { <corpo> }` (função com argumentos)
  - `define Nome Tipo` (declaração de novo tipo)
- Os tipos pré-definidos são os seguintes:
  - `int`
  - `float`
  - `string`
  - `bool`
  - `Tipo[IntExp]` (array com elementos do tipo `Tipo`)
  - `void` (tipo para funções sem valor de retorno - procedimentos)
- Os literais têm o formato "habitual":
  - Inteiros (`1; 30; 5000`)
  - Floats (`1.2; 0.1; .23; .22e-20`)
  - Strings (`"hello, world!"; "1.2"`)
  - Bools (`true; false`)

- Expressões binárias:
  - `+, -, *, / (int, float)`
  - `mod, ^ (int, float)`
  - `==, != (int, float, bool, string - e arrays)`
  - `<, >, <=, >= (int, float)`
  - `and, or (bool)`
- Expressões unárias:
  - Valor negativo (`-`)
  - Negação booleana (`not`)
- Afectações também são expressões:
  - `a = 1`
  - `a = b = c = 1`
  - `a[20] = b[i=2] = 3 - x`
- O corpo de uma função é constituído por *statements*. Statements podem ser:
  - Declarações (para variáveis locais);
  - Expressões (para afectações, outras expressões não produzem código "interessante" – mas podem ser aceites);
  - Instrução de retorno (`return Exp`)
  - Condicionais:
    - `if Booleano then { <corpo> }`
    - `if Booleano then { <corpo> } else { <corpo> }`
  - Ciclos (`while Booleano do { <corpo> }`)
- O corpo dos ciclos e condicionais é semelhante ao das funções.
- Um ciclo pode ser forçado a terminar com a instrução `break`, ou forçado a passar à próxima iteração, com a instrução `next` (equivalente ao `continue` do C ou Java).

Palavras e símbolos reservados

- `; " ( ) [ ] { } , : =`
- `+ - * / ^`
- `== < > <= >= !=`
- `mod and or not`
- `int float string bool void`
- `define if then else while do`
- `return break next`

Funções pré-definidas (parte da "biblioteca" do *Ya!*)

- `print(Exp)` → mostra o resultado de `Exp` no ecrã
- `input(lValue)` → guarda um valor escrito no teclado no `lValue` (tendo em conta o tipo do `lValue`). `lValue` pode ser qualquer "coisa" que seja válida do lado esquerdo de uma afectação (`a, a[1], a[i][j]`, etc).

Alternativas de implementação

- Como alternativa à geração de código MIPS, poderá escolher gerar código para outra arquitectura do seu agrado (ex: JVM, x86, LLVM, etc).

Omissões no enunciado

- Quaisquer detalhes de implementação omitidos neste enunciado ficam ao critério dos alunos, devendo as escolhas ser devidamente documentadas no relatório de implementação. (Ex: compatibilidade de tipos – `int/bool` em `if`s; *casting* implícito entre `int` e `float`; etc.)

Entrega

- O trabalho deve ser efectuado em "grupos" de 1 ou 2 alunos;
- O formato de entrega é um ficheiro compactado (preferencialmente .tar.gz) com uma directoria contendo todo o código do trabalho (numa subdirectoria /src) e respectivo Makefile, bem como um relatório de implementação (chamado relatorio_12345_12346.pdf, na directoria principal);
- O nome do ficheiro (e da directoria) tem de conter os números dos alunos que executaram o trabalho (exemplo: 12345-12346.tar.gz, que descompacta para a directoria 12345-12346);
- Apenas um dos elementos do grupo pode fazer a entrega do trabalho;
- A data limite para a entrega é 10/06/2020.