

Trabalho Prático de Compiladores¹

Prof. Thiago Borges de Oliveira

¹ Especificações do Trabalho da disciplina de Compiladores, 2015/1

Conforme previsto no plano de ensino da disciplina, $\frac{1}{3}$ da média refere-se a um trabalho que consiste em criar um projeto de compilador (analisador léxico, sintático e semântico) que será integrado ao *backend* do LLVM para gerar código executável para Arduino Uno R3. Adicionalmente, deverá ser implementado um programa na linguagem criada, o qual será transferido para o Arduino e movimentará um braço robótico (Figura 1).



Figura 1: Braço robótico.

O ALUNO (OU GRUPO) DEVE:

1. Especificar uma linguagem de programação, de alto ou baixo nível, para controlar o braço robótico. O braço deve ser usado para executar alguma tarefa, como por exemplo, pegar ou levantar objetos. Deve haver uma entrada no sistema. Os sensores disponíveis serão: um botão, dois fotoresistores e um microfone;
2. Implementar o analisador léxico e o analisador sintático da linguagem. Implementar tratamento de erro no analisador sintático;
3. Implementar o analisador semântico² e emissão de código usando classes em c++, as quais serão fornecidas e estão descritas a seguir. Essas classes fornecidas controlam as portas de entrada e as portas de saída do Arduino;
4. Criar um programa na linguagem para executar uma tarefa com o robô;
5. Fazer uma apresentação da linguagem e do robô executando a tarefa em sala de aula; e
6. Durante a apresentação, destacar as características da linguagem, as dificuldades encontradas no projeto e o aprendizado.

² Tabela de símbolos e árvore sintática.

Especificação das Classes de Emissão de Código

As seguintes classes podem ser instanciadas e anexadas na árvore sintática, para produzir o código do programa:

- `Int8(const char *n)`: Cria uma constante inteira de 8 bits, de valor *n*;
- `Int16(const char *n)`: Cria uma constante inteira de 16 bits, de valor *n*;
- `Int32(const char *n)`: Cria uma constante inteira de 32 bits, de valor *n*;
- `Float(const char *n)`: Cria uma constante float, de valor *n*;

- `Variable(const char *n, Node *e)`: Cria uma variável de nome *n*, com o valor da expressão *e*, que é um nó da árvore representando uma constante ou uma expressão aritmética;
- `Load(const char *n)`: Carrega o valor da variável *n* da memória, previamente criada com `Variable(n)`;
- `InPort(const char *p)`: Lê o valor da porta de entrada *p* do Arduino;
- `OutPort(const char *p, Node *e)`: Seta o valor da porta de saída *p* para o valor definido pela expressão ou constante *e*;
- `Delay(Node *mseg)`: Inclui uma espera de *mseg* milissegundos. *mseg* pode ser uma constante ou expressão aritmética;
- `BinaryOp(Node *l, char op, Node *r)`: Realiza uma operação binária (+, -, *, /) com os operandos *l* e *r*. *l* e *r* podem ser constantes ou expressões aritméticas.
- `CmpOp(Node *l, yytokentype op, Node *r)`: Realiza a operação de comparação *op*, entre os operandos *l* e *r*. O operador *op* deve ser definido na linguagem léxica conforme a Tabela 1. *r* e *l* são constantes ou expressões aritméticas.
- `If(Node *e, Node *then, Node *else)`: Cria uma condição no programa, que avalia a expressão booleana *e* e desvia para *then* ou *else* conforme o resultado. Este é o comando *if* da linguagem.
- `While(Node *e, Node *stmts)`: Cria um laço de repetição no programa, que executa os comandos em *stmts*, enquanto a expressão *e* é avaliada como verdadeira. A expressão *e* deve ser uma instância da classe `CmpOp`. *stmts* deve ser do tipo *Stmts*.
- `Stmts(Node *ss)` ou `Stmts(Node *ss, Node *s)`: Classe que representa um comando ou um bloco de comandos, ou seja, um conjunto de instâncias das classes acima descritas. Deve ser usada em uma regra recursiva, semelhante a regra da Lista 1.
- `Program().generate(Node *n)`: Classe e método final que gera o código fonte intermediário. Pode ser chamada no símbolo inicial da gramática. *n* deve ser do tipo *Stmts*.

Documentação do Projeto de Hardware

Segue abaixo o esquemático (Figura 2) e o projeto em *protoboard* (Figura 3) do *hardware* que será disponibilizado. Você pode realizar alterações no projeto, mas considere implementar inicialmente um compilador para esta especificação.

O *hardware* possui as entradas MIC1, S1, R7 e R8, ligadas no Arduino, conforme especificado na Tabela 2. Existem 12 saídas, especificadas na Tabela 3.

Tabela 1: Tokens e significado dos operadores de comparação.

Token	Operador
EQ_OP	==
NE_OP	!=
GE_OP	>=
LE_OP	<=
GT_OP	>
LT_OP	<

Lista 1: Exemplo de produções para *Stmt*.

```
stmts : stmts stmt {
        $$ = new Stmts($1, $2);
    }
    | stmt {
        $$ = new Stmt($1);
    }
    ;
```

Tabela 2: Descrição das entradas do *hardware*.

Saída	Descrição
A0	Nível de luz no fotoresistor R7
A1	Nível de luz no fotoresistor R8
A2	Sinal do Botão S1
A3	Sinal do Microfone MIC1

Tabela 3: Descrição das saídas do *hardware*.

Saída	Descrição
D3	Liga LED3
D5	Liga LED2
D6	Liga LED1
D0	Liga Motor 1 sentido horário
D1	Liga Motor 1 sentido anti-horário
D2	Liga Motor 2 sentido horário
D4	Liga Motor 2 sentido anti-horário
D7	Liga Motor 3 sentido horário
D8	Liga Motor 3 sentido anti-horário
D9	Liga Motor 4 sentido horário
D10	Liga Motor 4 sentido anti-horário
D11	Liga Motor 5 sentido horário
D12	Liga Motor 5 sentido anti-horário

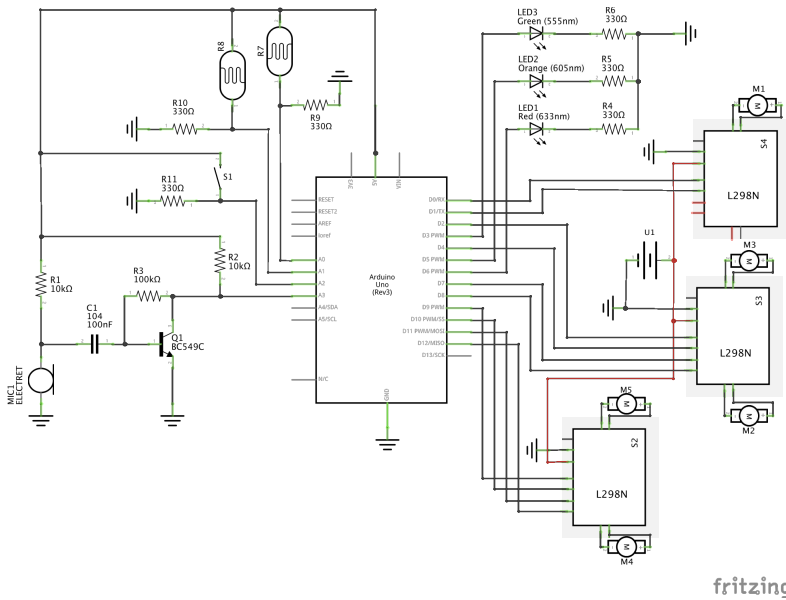


Figura 2: Esquemático do projeto de *hardware*.

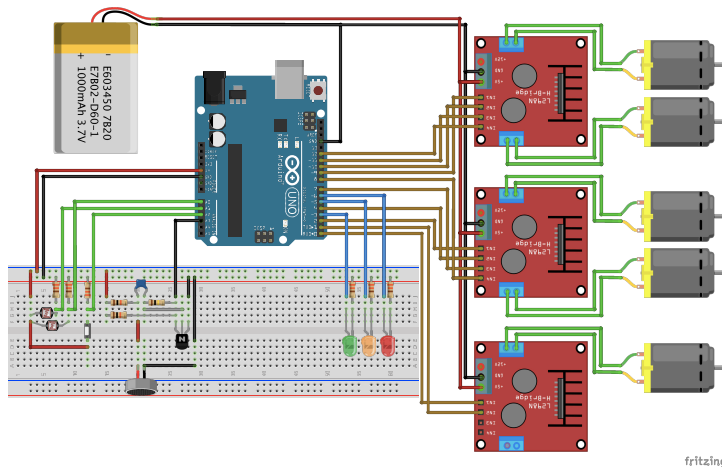


Figura 3: Projeto de *hardware* na protoboard.

Robcmp: Exemplo de Linguagem de Baixo Nível

O programa a seguir está escrito em uma linguagem de baixo nível, que possui comentários, atribuições, expressões aritméticas, condições simples (sem *and* e *or*) e laço de repetição. Esta linguagem pode ser usada para controlar o robô, porém é de propósito geral, ou seja, qualquer *hardware* ligado no Arduino poderia ser controlado com ela³.

```
if (in0 > 10)      /* lê porta 0 e verifica se é > 10 */
    out3 = 255;    /* seta porta 3 para 255 */

i = 1 / 1 + 0;     /* cria variável i com expressão aritmética */
while (i < 10) {   /* repete enquanto i < 10 */
    i = i + 1;     /* aumenta o valor de i */
}

if (in1 - 10 > 10) /* testa porta 1 */
    out6 = 255 - i; /* seta porta 6 para 255 - i */
else
    out6 = 0;       /* seta porta 6 para 255 */
```

³ O código completo desta linguagem e compilador está disponível em <http://github.com/thborges/robcmp>.