C - Charles

Integrantes

- Giovanni Morassi
- Henrique Finatti
- Tiago Fagundes
- Mateus Marana

Como rodar?

O projeto é feito utilizando o gerenciador de pacotes UV, para rodar como o esperado é necessário ter instalado em sua maquina ou então, criar um ambiente virtual e instalar o rich com o seu gerenciador de pacotes (ele é a unica dependencia).
Caso prefira, é possivel rodar com o python global caso tenha o rich instalado

Caso você ja tenha o UV instalado no seu computador, basta entrar pelo terminal
na pasta onde contem o arquivo pyproject.py e rodar o comando

```
uv run src/main.py <caminho_arquivo>
```

Existem a flag `--limpo` e `--no-fpc`, a limpo roda e traz o resultado final do código compilado e a no fpc, vai apenas traduzir e não gerará nenhum código em pascal.

```
uv run src/main.py <caminho_arquivo> --no-fpc
```

```
uv run src/main.py <caminho_arquivo> --limpo
```

e caso deseja utilizar a nossa IDE, bastar rodar o comando

```
uv run IDE/ide.py
```

ela iniciará uma interface grafica, onde será possivel analizar a arvore, código em pascal e a saida!

Caso deseje instalar o uv o basta clickar no seguinte link: UV Astral

O projeto é desenvolvido em python utilizando ferramentas para tipagem e linting.

Utilizamos o ruff (formater e linter) e pyright (type checker). Eles nos trazem um padrão com o mercado.

Dependencias

O projeto tem o intuito de ser o mais puro possivel, utilizando apenas o rich, para a criação de prints coloridos, apenas para questão estética.

Lista de Tokens

> Abaixo estão todos os tokens aceitos pelo analizador lexico.

- **id** : [a-zA-Z][a-zA-Z0-9]*
- **palavras_reservadas**: (string | boolean | float | int | for |while | if | else | else if | input | console)
- **operador_atribuicao**: "<-"
- **operador_matematico**: ("+" | "-" | "/" | "*" | "%")
- **operador_logico**: ("&&' | "||" | "!")
- **operador_relacional**: (">" | "<" | ">="| "<=" | "=" | "!=")
- **Operador saida**: "<<"
- **abre_parenteses**: "("
- **fecha_parenteses**: ")"
- **abre_chaves**: "{"
- **fecha_chaves**: "}"
- **fim_instrucao**: ";"
- **numero_inteiro**: [0-9]+
- **numero_decimal**: [0-9]+.[0-9]+
- **textos**: [a-z _ A-Z 0-9]*

Gramatica

> Exemplos da sintaxe da nossa linguagem
> obs: Linguagem livre de contexto ainda não feita, está em nossos proximos steps

- **Input**

```
int variavel;
variavel << input;
```

- **Console output**

```
console << variavel;
```

- **Controle de fluxo**

```
int variavel;
variavel << input;
if (variavel > 10) ->{
  console << variavel
} else ->{
  console << "Esta dentro do intervalo -10 e 10";
}
```

- **Atribuição**

```
int variavel <- 30;
variavel <- variavel + 3;
console << variavel;
```

- **Operadores logicos**

```
int variavel;
variavel << input;
if ( variavel > 10 && variavel < 30 ) -> {
  estaNoIntervalo = true;
}
if(variavel = 30){
  console<< "A variavel é exatamente 30";
} else -> {
  console<< "A variavel é diferente de 30";
}
```

Gramática livre de contexto

- **programa** -> tipo 'charles' '()' '->' '{' bloco '}'
- **bloco** -> cmd bloco_options
- **bloco_options** -> bloco | e
- **cmd** -> leitura ';' | escrita ';' | atribuicao ';' | cmd_if | declarar;| loop_for | loop_while
- **declarar** -> tipo declarar_options
- **declarar_options** -> id declarar_options_linha ';'
- **declarar_options_linha** -> '<-' atribuicao_options | e
- **cmd_if** -> 'if' '(' condicional ')' '->' '{' bloco '}' option_else
- **option_else** -> 'else' '->' '{' bloco '}' | e
- **condicional** -> expressão operador_relacional expressão condicional_linha |'!' expressão condicional_linha
- **condicional_linha** -> operador_logico condicional condicional_linha
- **expressão** -> exp_prioridade expressão_linha
- **expressão_linha** -> '+' exp_prioridade expressão_linha | '-' exp_prioridade expressão_linha | e
- **exp_prioridade** -> fator exp_prioridade_linha
- **exp_prioridade_linha** -> '*' fator exp_prioridade_linha | '/' fator exp_prioridade_linha | e
- **fator** -> id | num | '(' expressão ')'
- **leitura** -> id '<<' 'input'
- **escrita** -> 'console' '<<' escrita_options
- **escrita_options** -> id | texto_string
- **atribuicao** -> id '<-' atribuicao_options
- **atribuicao_options** -> expressão | texto_string
- **texto_string** -> ""[a-zA-Z 0-9]""
- **num** -> num_int | num_decimal
- **operador_relacional** -> '>' | '<' | '>=' | '<=' | '=' | '!='
- **tipo** -> 'int'| 'float' | 'boolean' | 'string'
- **operador_logico** -> '&&' | '||' | '!'
- **id** -> [a-zA-Z][a-zA-Z0-9]*
- **num_int** -> [0-9]+
- **num_decimal** -> [0-9]+.[0-9]+
- **loop_for** -> 'for' '(' declarar ';' condicional ';' atribuicao ')' '->' '{' bloco '}'
- **loop_while** -> 'while' '('condicional ')' '->' '{' bloco '}'