Disciplina: Compiladores

Professora: Dr. Anna Paula Rodrigues

Alunos: Luccas Castro de Souza e Emanuel Catão Montenegro

Trabalho Prático 1

A linguagem que propomos é projetada especificamente para a criação rápida e facilitada de interfaces gráficas simples e jogos 2D básicos. O o**bjetivo** aqui é **fornecer uma plataforma acessível para aqueles que desejam prototipar interfaces gráficas sem se preocupar com a complexidade das bibliotecas gráficas de baixo nível** (em nosso caso, a GTK). Trata-se, portanto, de uma **DSL (linguagem de domínio específico)**; as razões para essa classificação já foram devidamente apresentadas.

Bibliotecas gráficas C tradicionais, como GTK e SDL têm uma curva de aprendizado consideravelmente íngreme, especialmente para iniciantes ou programadores não familiarizados com algumas nuances da programação em C. O objetivo da linguagem é atacar esse problema pontualmente. Boa parte do **público-alvo** já foi apresentado (**os iniciantes e aqueles que não queiram ter que lidar com a complexidade das bibliotecas gráficas**), mas essa linguagem pode ser também útil para **professores que desejam ensinar conceitos de programação e design de interfaces de maneira mais facilitada**. Das necessidades e expectativas com o projeto, por primeiro objetivamos alcançar a facilidade de uso e sintaxe intuitiva.

Pensamos a linguagem com **paradigma imperativo e procedural**. O foco está na execução sequencial de comandos para criar e manipular elementos gráficos. Esperamos valer-se ainda de conceitos de programação baseada em eventos, permitindo que ações sejam desencadeadas em resposta a interações na interface (como clicar em algo ou teclar algo). A seguir apresentamos alguns elementos da semântica.

Semântica

- **`<statement_list> ::= <statement> | <statement> <statement_list>`**
  - **Descrição**: É uma lista de declarações, onde cada declaração pode ser uma instrução (statement) e a lista pode conter várias instruções em sequência.
  - **Semântica**: Permite a execução sequencial de comandos. Cada instrução é processada uma após a outra.
  - **Exemplo**: Um código que contém múltiplas instruções, como `let score = 0;` seguido de `move_element(...)`, será executado na ordem em que aparece.
- **`<statement> ::= <declaration>`**

  `| <assignment>`

  `| <function_call>`

  `| <selection_statement>`

  `| <loop_statement>`

  `| <repeat_statement>`

  `| <move_element>`

  `| <add_element>`

  `| <on_keypress>`

  `| <close_window>`
  - **Descrição**: Um statement é tudo aquilo que executa uma ação específica.
- **`<identifier> ::= [a-zA-Z_][a-zA-Z0-9_]*`**
  - **Descrição**: Define a forma que um identificador pode ser escrito
  - **Semântica**: Um identificador deve começar por caracteres alfabético ou _ e pode seguido de um ou mais caracteres alfanuméricos além do _.
  - **Exemplo**: `score;` é um identificador válido. `2core` é um identificador inválido.
- **`<expression> ::= <literal> | <identifier> | <binary_operation> | <function_call>`**

    - **Descrição**: Calcula um valor a partir de literais, variáveis, operações binárias e afins
    - **Semântica**: Uma expressão é avaliada para produzir um valor. Operações binárias, como soma ou multiplicação, combinam os valores de duas sub-expressões. O valor resultante de uma expressão pode ser atribuído a uma variável, usado como argumento de uma função, ou avaliado dentro de uma estrutura de controle.
    - **Exemplo**: `x + y` soma os valores de `x` e `y`.
- **`<literal> ::= <number> | <string>`**
    - **Descrição**: Definição de um literal
    - **Semântica**: Um literal pode ser um número ou string.
    - **Exemplo**: `score = 10;` 10 é um literal numérico
- **`<operator> ::= "+" | "-" | "*" | "/" | "and" | "or" | ">"`**

  `| "<" | "<=" | "=<" | "!="`
    - **Descrição**: Operadores suportados (aritméticos, lógicos e relacionais)
- **`<binary_operation> ::= <expression> <operator> <expression>`**
    - **Descrição**: Definição de uma operação binária
    - **Semântica**: Uma operação binária é composta de duas expressões que são avaliadas com o uso de um operador.
    - **Exemplo**: `score = 10 + 10;` A operação binária ao lado é composta de duas expressões literais e um operador aritmético.
- **`<number> ::= [0-9]+`**
    - **Descrição**: Definição de number
    - **Semântica**: Um number é uma sequência de um ou mais algarismos
    - **Exemplo**: `score = 10;` 10 é um number.
- **`<string> ::= '"' .* '"'`**
    - **Descrição**: Definição de uma cadeia de caracteres
    - **Semântica**: Uma cadeia de caracteres é delimitada por aspas. É qualquer sequência de zero ou mais caracteres.
    - **Exemplo**: `score = “”;` score recebeu um literal de string vazia.

Declarações e Atribuições

- **`<declaration> ::= "let" <identifier> "=" <expression> ";"`**

    - **Descrição**: Declara uma nova variável e a inicializa com o valor da expressão fornecida.
    - **Semântica**: Quando uma variável é declarada, a memória é alocada para armazenar o valor resultante da expressão. As variáveis são mutáveis.
    - **Exemplo**: `let score = 0;` cria uma variável `score` e a inicializa com o valor `0`.
- **`<assignment> ::= <identifier> <assignment-operator> <expression> ";""`**
    - **Descrição**: Faz uma atribuição a uma variável já declarada.
    - **Semântica**: Tendo uma variável declarada, podemos atribuir um valor a ela.
    - **Exemplo**: `score = 0;` atribui a variável `score` com o valor `0`.
- **`<assignment-operator> ::= =`**
  `| *=`
  `| /=`
  `| %=`
  `| +=`
  `| -=`
    - **Descrição**: Operadores de atribuição suportados na linguagem.
    - **Semântica**: Tendo uma variável declarada e um valor, executa a operação especificada pelo operador unitário evidente à esquerda da igualdade. Para fins de simplicidade, daremos suporte a apenas os operadores matemáticos comuns

Funções

- **`<function_call> ::= function <identifier> "(" <argument_list> ")"`**
    - **Descrição**: Invoca uma função com os argumentos fornecidos.
    - **Semântica**: Quando uma função é chamada, os argumentos são avaliados e passados para a função. A função executa seu corpo de código. Por questão de simplicidade, imaginamos a primeiro momento trabalhar com funções sem retorno algum.
    - **Exemplo**: `function center_element(element);` centraliza o elemento.

- **`<argument_list> ::= <expression> | <expression> "," <argument_list>`**
    - **Descrição**: A lista de argumentos refere-se aos valores efetivos que são passados para uma função quando ela é chamada. .
    - **Semântica**: São os **valores** atribuídos aos parâmetros definidos.
    - **Exemplo**: Ao chamar `add(5, 10)`, `5` e `10` são os **argumentos**
- **`<parameter_list> ::= <identifier> | <identifier> "," <parameter_list>`**
    - **Descrição**: Lista de parâmetros, é a definição de parâmetros dentro de uma função.
    - **Semântica**: Serve como placeholders que esperam receber valores durante a execução.
    - **Exemplo**: `function center_element(element);` element é um parâmetro.

Estruturas de Seleção

- **`<selection_statement> ::= if (<condition>) { <statement_list> } | if (<condition>) { <statement_list> } else { <statement_list> }`**
    - **Descrição**: Executa um bloco de código se a condição for verdadeira, caso contrário, executa o bloco `else`.
    - **Semântica**: A condição é avaliada. Se o resultado for verdadeiro, o bloco de código associado é executado; caso contrário, o bloco `else` (se presente) é executado. Se a condição não for booleana, é levantada uma exceção.
    - **Exemplo**: `if (score > 100) { print("High score!"); }` Imprime "High score!" se `score` for maior que 100.

Estruturas de Repetição

- **`<repeat_statement> ::= "repeat" <number> "times" "{" <statement_list> "}"`**
    - **Descrição**: Executa o bloco de código um número determinado de vezes.

    - **Semântica**:O bloco de código é executado repetidas vezes, de acordo com o número informado de repetições.
    - **Exemplo**: `repeat 5 times { print("Oi"); }` imprime na tela a string "Oi" cinco vezes.
- **`<loop_statement> ::= "loop" "{" <statement_list> "}" "every" <number> "ms"`**
    - **Descrição**: Define um loop que é executado continuamente em intervalos de tempo definidos (tempo dado em milissegundos).
    - **Semântica**:O bloco de código é executado repetidas vezes continuamente.
    - **Exemplo**: `loop { print("Oi"); } every 500 ms` imprime na tela a string "Oi" a cada 500 milissegundos.

Manipulação de Elementos Gráficos

- **`<create_window> ::= "create window" "(" <string> "," "width=" <number> "," "height=" <number> ")"`**
    - **Descrição**: Cria uma nova janela com o título e as dimensões especificadas.
    - **Semântica**: Inicializa a interface gráfica, criando uma nova janela que será usada como o contêiner principal para todos os elementos gráficos. A janela é automaticamente exibida após sua criação.
    - **Exemplo**: `create window("My Game", width=800, height=600);` cria uma janela de 800x600 pixels com o título "My Game".
- **`<add_element> ::= "add element" "(" <identifier> "," "type=" <element_type> "," <attributes> ")"`**
    - **Descrição**: Adiciona um novo elemento gráfico à interface, como um botão, imagem, ou rótulo de texto.
    - **Semântica**: O elemento gráfico é criado com o tipo e os atributos especificados e é adicionado à janela ou contêiner atual. Cada elemento é identificado por um nome único, que pode ser usado para manipulá-lo posteriormente.
    - **Exemplo**: `add element("startButton", type="button", text="Start");` adiciona um botão com o texto "Start".
- **`<element_type> ::= "sprite" | "label" | "textbox" | "button"`**
    - **Descrição**: Tipos de elementos suportados.

    - **Semântica**: Sprite é objeto de imagem gráfica, Label é um elemento de texto simples, Textbox é um elemento de caixa de texto interativa e button um elemento que define um botão
- **`<attributes> ::= <attribute> | <attribute> "," <attributes>`**
    - **Descrição**: Listagem de um ou mais atributos separados por vírgula
- **`<attribute> ::= "image=" <string> | "text=" <string> | "placeholder=" <string> | "hidden=" <boolean> | "x=" <number> | "y=" <number>`**
    - **Descrição**: Atributos suportados na linguagem
    - **Semântica**: image recebe uma string de path da imagem, text recebe o texto a ser exibido no elemento de label, placeholder recebe a string a ser exibida como placeholder no textbox, hidden especifica se o elemento será ou não visível (por padrão é true) e x e y especificam as coordenadas do elemento na tela.
    - **Exemplo:** `add element("startButton", type="button", text="Start")`o adiciona o elemento startButton, de tipo botão e que tem o texto Start exibido nele.
- **`<move_element> ::= "move element" "(" <identifier> ")" "to" "(" "x=" <number> "," "y=" <number> ")"`**
    - **Descrição**: Move o elemento gráfico especificado para a posição (x, y).
    - **Semântica**: O elemento identificado pelo nome é reposicionado na janela ou contêiner pai. As coordenadas x e y especificam a nova posição relativa ao canto superior esquerdo da janela.
    - **Exemplo**: `move element("player", x=100, y=200);` move o elemento `player` para a posição (100, 200).
- **`shift element("<identifier>", x=<number>, y=<number>);`**
    - **Descrição**: Desloca o elemento gráfico especificado para a posição uma quantidade definida de pixels, dadas por x e y.
    - **Semântica**: O elemento identificado pelo nome é reposicionado na janela ou contêiner pai. X e Y especificam os deslocamento horizontal e vertical do elemento em relação ao ponto em que está situado.
    - **Exemplo**: `shift element("player", x=100, y=200);` desloca o elemento `player` 100 pixels para a direita e 200 pixels para cima.
- **`<show_element> ::= "show element" "(" <identifier> ")"`**
    - **Descrição**: Torna visível um elemento que estava oculto

    - **Semântica**: Exibe um elemento já existente na interface
    - **Exemplo**: `show element("player");` exibe o elemento player.
- **`<hide_element> ::= "hide element" "(" <identifier> ")"`**
    - **Descrição**: Torna oculto um elemento que estava invisível
    - **Semântica**: Oculta um elemento já existente na interface
    - **Exemplo**: `hide element("player");` oculta o elemento player.

Eventos

- **`<on_keypress> ::= "on keypress" "(" <key> ")" "{" <statement_list> "}"`**
    - **Descrição**: Define um comportamento específico a ser executado quando determinada tecla é pressionada
    - **Semântica**: O comando monitora o pressionamento das teclas e executa o conjunto de declarações delimitadas pelas chaves
    - **Exemplo**:

```
on keypress("A") {
    move element("player", x=-10, y=0);
}.
```

- **`<key> ::= "left arrow" | "right arrow" | "space" | <custom_key>`**
    - **Descrição**: Especifica a tecla que será monitorada pelo evento de key press
    - **Semântica**: Exclui um elemento, liberando os seus recursos e retirando-o da interface
- **`<custom_key> ::= <string>`**
    - **Descrição**: Especifica uma tecla personalizada definida pelo usuário
    - **Semântica**: Exclui um elemento, liberando os seus recursos e retirando-o da interface
- **`<on_click> ::= "on click" "(" <identifier> ")" "{" <statement_list> "}"`**
    - **Descrição**: Monitora quando um elemento é clicado.
    - **Semântica**: Quando o elemento especificado é clicado, executa o conjunto de declarações do bloco
    - **Exemplo:**

```
on click("bird") {
      shift element("bird", x=0, y=10);
}.
```