UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA
CIÊNCIA DA COMPUTAÇÃO

GUSTAVO COSTA DUARTE
LEONARDO NASCIMENTO DOS SANTOS
VINÍCIUS BERGER

DESENVOLVIMENTO DE UM INTERPRETADOR DO SISTEMA LINDENMAYER

VITÓRIA
2015

GUSTAVO COSTA DUARTE
LEONARDO NASCIMENTO DOS SANTOS
VINÍCIUS BERGER

DESENVOLVIMENTO DE UM INTERPRETADOR DO SISTEMA LINDENMAYER

Trabalho apresentado à disciplina Estrutura de Dados I, do curso superior em Ciência da Computação da Universidade Federal do Espírito Santo, como requisito para obtenção da avaliação na disciplina.
Professor:Thomas Walter Rauber.

VITÓRIA
2015

1 RESUMO

O script criado lê um arquivo de extensão tipo “*.lsy”*, que contém as instruções de entrada para a inserção e manipulação dos dados no sistema e gera dois arquivos de saída Postscript com o desenho formado de acordo com as especificações do sistema Lindenmayer.

2 INTRODUÇÃO

Um sistema de Lindenmayer é uma gramatica formal que permite a geração de palavras finais (strings) que podem ser interpretadas como comandos simples de um sistema gráfico de tartaruga. No sistema Lindenmayer existe um axioma e uma regra de formação. A aplicação da regra um determinado numero de vezes gerará uma string final. A sintaxe adotada neste trabalho é a mesma do material de Jennings, com algumas poucas modificações, como será exposto mais adiante.

O presente trabalho consiste em ler um arquivo “*.lsy*” e armazenar cada parâmetro em uma variável específica. Posteriormente, utilizar esses dados para a formação de uma árvore que armazenará em cada nível uma ordem de iteração da regra de produção. O último nível, ou seja, as folhas da árvore, representa a string final que será utilizada para gerar os arquivos Postscript.

O projeto de um programa engloba, entre outras, a fase de identificação das propriedades dos dados e suas características funcionais. Uma representação adequada dos dados, em vista das funcionalidades que devem ser atendidas, constitui uma etapa fundamental para a obtenção de programas eficientes e confiáveis.

Estruturas lineares como vetores e listas nem sempre são as mais adequadas para representar os dados do sistema, principalmente nas situações em que eles precisam ser dispostos de maneira hierárquica. Nesse caso, a melhor opção é utilizar a estrutura de dados conhecida por “árvore”.

Uma árvore é uma estrutura de dados que pode ser representada como uma hierarquia onde cada elemento é chamado de nó. O nó inicial ou o primeiro elemento é chamado de raiz. As árvores são definidas de acordo com o número máximo de filhos que possuem. Nesse sentido, árvores binárias têm no máximo dois filhos em cada nó, ou seja, duas sub-árvores filhas. Um nó sem filhos é chamado de folha.

Dessa forma, visando atender o objetivo do projeto, desenvolveu-se um software baseado na linguagem de alto nível C, com o uso de tipos abstratos de dados, árvores com número variável de filhos, pilhas e alocação dinâmica de memória. Ferramentas como o *Notepad*++ e o *Gedit* foram utilizadas para a criação e edição do código, enquanto a ferramenta *Valgrind* foi utilizada para a verificação de vazamento de memória.

3 OBJETIVOS

Criação de um interpretador do sistema de Lindenmayer (L-system), com algumas funcionalidades reduzidas, conforme apresentado na próxima seção. Visualização do objeto gráfico gerado. Representação e manipulação de informação estruturada por linguagem de programação de alto nível: listas, pilhas e árvores.

4 METODOLOGIA

A primeira função desenvolvida para implementação do sistema foi a de leitura de arquivo. Essa função recebe como parâmetro o ponteiro de algumas variáveis declaradas na função principal e uma cópia nome escolhido pelo usuário para abertura do arquivo de entrada.

Posteriormente, de posse de todas as informações contidas no arquivo de entrada, iniciou-se a implementação das funções que utilizarão a estrutura da árvore para

armazenar os níveis das iterações, de acordo com a ordem, axioma e regra de produção lidos anteriormente. O objetivo de utilizar essa estrutura é que a concatenação das folhas representará a string final.

Assim, a próxima funcionalidade acrescentada foi a de converter a string final gerada de acordo com as especificações do sistema de Lindenmayer para um formato que pudesse ser interpretado pelo Postscript, com auxílio da interface disponibilizada pela especificação do trabalho. Dessa forma, as duas strings geradas anteriormente são utilizadas para dar origem a dois objetos gráficos em formato Postscript.

Vale ressaltar que o espaço ocupado na memória pelas strings são alocados dinamicamente, a fim de permitir um alto desempenho caso a ordem de iteração seja elevada e reduzir o consumo de memória caso haja poucas iterações. A ferramenta Valgrind foi utilizada a fim de verificar a rigidez na administração da memória alocada dinamicamente pelas funções.

Por fim, a metodologia de programação baseada em tipos abstratos de dados foi implementada para encapsular a implementação das estruturas e operações relacionadas a elas. Além disso, diversos modelos de arquivos-entrada foram desenvolvidos a fim de contemplar todos os possíveis erros do sistema.

As ferramentas utilizadas no desenvolvimento do projeto estão listadas a seguir:

- Linguagem de alto nível C;
- Sistema operacional Linux Mint;
- Compilador gcc (GNU Compiler Collection);
- Depurador Valgrind;
- Bibliotecas padrão stdlib.h, stdio.h e string.h
- Editores Notepad++ e Gedit.

4.1 ESTRUTURA

Os dados que serão manipulados dentro do sistema estão listados na tabela a seguir.

Preâmbulo	
angle n	Vira a tartaruga 360/n graus quando o comando + ou - for acionado.
order n	Aplique a regra de produção n vezes.
rotate α	Rotacione a imagem inteira α antes de começar a desenhar.
Produção	
Axiom string	A string inicial do L-Sistema (ordem 0).
p = string	Regra de produção. Em cada iteração (ordem) todas as instâncias do símbolo único p serão substituídos pela string “string”. Note que p obrigatoriamente é um único símbolo, e não pode ser do conjunto de caracteres especiais =,+,−,!,|,[,],<,>,@,/,\, ,c, ou espaço. Maiúsculos são diferentes de minúsculos.
Gráficos de Tartaruga	
F	Mover a tartaruga com caneta baixada.
G	Mover a tartaruga com caneta levantada.
+	Virar a tartaruga por ângulo positivo.
-	Virar a tartaruga por ângulo negativo.
[	Empilhar o estado da tartaruga (posição e orientação).
]	Desempilhar o estado da tartaruga.

4.1.1 REPRESENTAÇÃO DA INFORMAÇÃO

Todas as produções foram organizadas em uma árvore com número de filhos variável. A string final são as folhas dessa árvore e é composta unicamente pelos comandos gráficos da tartaruga da tabela anterior, sem serem interpretados pelo sistema de produção da string final. Exemplo: Com o axioma ’++FX’ e a regra ’X→[-FX]+FX’, com ordem 1, a string final deve ser ’++F[-FX]+FX’; com ordem 2 a string final deve ser ’++F[-F[-FX]+FX]+F[-FX]+FX’.

Os comandos de empilhar e desempilhar o estado da tartaruga “[“ e ”]”, são interpretados do lado do sistema não gráfico. Para atingir este objetivo existe uma pilha que guarda uma estrutura do estado atual da tartaruga (posição e orientação). Além disso, os comandos de desenhar uma reta (F) e se mover sem desenhar (G) são quebrados em unidades elementares, pois o retorno para um estado anterior requer a inicialização do caminho atual (newpath do Postscript).

Dessa maneira, a string final gerada é convertida para uma string de tartaruga composta por comandos básicos com a seguinte sintaxe:

n x0 y0 m x1 x2 l s

Onde:

- (x0,y0) e (x1,y1) são os pontos iniciais e finais de uma reta;
- n significa newpath;
- l significa lineto;
- m significa moveto;
- s significa stroke.

Esses comandos são diretamente interpretados pelo Postscript.

Dessa forma, encontrando o símbolo +, a orientação é incrementada pelo ângulo base; - decrementa. Quando aparecer [, o estado atual é empilhado; com] é

desempilhado, ou seja, é restabelecida a posição e orientação. Quando aparecer F, um comando básico da tartaruga é acrescentado à string da tartaruga. A nova posição é calculada da seguinte maneira:

$$x1 = x0 + z \cos(\alpha),\ y1 = y0 + z \sin(\alpha)$$

Onde z é o comprimento da reta (stroke length) e α é a orientação atual.

4.1.2 INTERFACE

Toda a interação funciona através de arquivos de entrada e saída. O usuário apenas escolhe qual arquivo deseja usar como entrada de dados e escolhe dois nomes para os arquivos Postscript de saída. A sintaxe do arquivo de entrada é a seguinte:

angle <Ângulo> ; <Comentários>
order <Ordem>
axiom <Axioma>
<Instância> = <Regra de Produção>

Os elementos podem estar dispostos em qualquer ordem, desde que estejam cada um em uma linha e todos os elementos estejam presentes no arquivo de entrada. Os comentários podem ser acrescentados ao final de cada linha, precedidos por um ponto-e-vírgula (;).

4.2 FUNCIONAMENTO

Inicialmente o programa importa as bibliotecas necessárias para se trabalhar com entradas e saídas de dados (stdio.h), alocação de memória (stdlib.h), manipulação de cadeia de caracteres (string.h) e funções matemáticas (math.h), além dos arquivos de protótipo (tad.h, psinterface.h e lsystem.h) dos módulos (tad.c, psinterface.c e lsystem.c).

Ao iniciar, o programa inicializa as variáveis que comportarão as informações necessárias ao sistema. Logo em seguida é solicitado que o usuário informe o nome do arquivo de entrada do qual se deseja importar os dados, além de dois nomes para arquivos de saída que armazenarão o código Postscrip, ou seja, os objetos gráficos.

O software, então realiza uma chamada à função de leitura de arquivo, utilizando como parâmetro o nome do arquivo de entrada informado anteriormente e vários ponteiros para variáveis que foram declaradas na função principal. A partir desse ponto as informações contidas no arquivo de entrada estarão à disposição do sistema.

Uma nova variável do tipo “tpÁrvore” é declarada e a raiz da árvore é criada, com a informação obtida da variável “subst”, que comporta a instância a ser substituída nas iterações da regra de produção.

Para criar o primeiro nível da árvore, ou seja, a ordem zero (axioma), é utilizado um laço de repetição que varia de acordo com o tamanho da string que comporta o axioma, guardando cada caractere em uma sub-árvore da raiz.

Outro laço de repetição é então utilizado para invocar a função que cria mais um nível na árvore, de acordo com o valor armazenado na variável “order”. Assim, se o valor de ordem for cinco, por exemplo, a função que cria mais um nível será chamada cinco vezes.

O próximo passo consiste em alocar espaço na memória para armazenar a string final, que é composta pelos caracteres armazenados nas folhas da árvore. Para isso é invocada uma função que conta quantas folhas existem. Essa quantidade é utilizada para calcular o espaço de memória necessário. Uma função que forma a string final é chamada para percorrer as folhas da árvore e armazenar as informações no espaço alocado.

De posse da string final é possível alocar espaço para a string do Postscript. Basta contar a quantidade de ocorrências de caracteres iguais a ‘F’ ou ‘G’ na string final e

utilizar esse valor multiplicado pelo tamanho da string Postscript de cada comando da tartaruga para alocar o espaço. Uma função que converte a string final em string Postscript é invocada para preencher a variável correspondente.

A fim de apresentar ao usuário todo o processamento realizado, uma função do tipo “void” imprime todas as variáveis lidas no arquivo de entrada, além da árvore gerada.

Por fim, as duas strings geradas anteriormente são utilizadas como parâmetros das duas funções disponibilizadas pela especificação do trabalho. Essas funções geram os arquivos Postscrip referentes aos objetos gráficos.

4.3 ERROS TRATADOS

O tratamento dos erros acontece dentro da função que lê um arquivo. São tratados os seguintes erros:

- Arquivo de entrada não localizado;
- Valor de ângulo não informado ou inconsistente;
- Valor de ordem não informado ou inconsistente;
- Falta de alguma informação necessária.

5 RESULTADOS E AVALIAÇÃO

Foram realizados testes utilizando o depurador Valgrind com objetivo principal de verificar se o software produzido está administrando a memória alocada de maneira correta. Os resultados foram positivos tanto para casos de sucesso quanto para os diversos erros que possivelmente aconteçam durante a execução. Diversos arquivos de entrada, com dados estrategicamente errados foram utilizados a fim de se verificar a “desenvoltura” do software.

Um erro identificado, mas não tratado neste projeto é o seguinte:

- Na leitura de um dos parâmetros (axiom), a função de leitura irá procurar uma string após a identificação do parâmetro. Caso ela não encontre na mesma linha essa string, o sistema pega a próxima linha como sendo o axioma. Isso não deveria ocorrer. Na verdade, deveria ser emitido um erro de leitura.

Conclui-se que o programa cumpriu o objetivo de realizar a leitura dos dados presentes no arquivo de entrada, relatar possíveis erros encontrados na sintaxe, utilizar as estruturas de dados adequadas para representar os dados hierarquicamente (árvores) e gerar os objetos gráficos desejados.

6 REFERÊNCIAS

CELES, Waldemar; CERQUEIRA, Renato; RANGEL, José Lucas. **Introdução a estruturas de dados**: com técnicas de programação em C. 11ª triagem. Rio de Janeiro: Elsevier, 2004.