IF61C—Fundamentos de Programação 1

Projeto Computacional - Dragon Curves

1 Dragon Curves

O objetivo deste projeto computacional é implementar funções que auxiliem no desenho de um tipo de fractal conhecido como "Dragon Curves". Ele é gerado da seguinte forma: iniciado a partir de um segmento base, deve-se substituir cada segmento por outros dois segmentos com um ângulo agudo e uma rotação, alternando da direita para esquerda. A figura abaixo ilustra o processo:

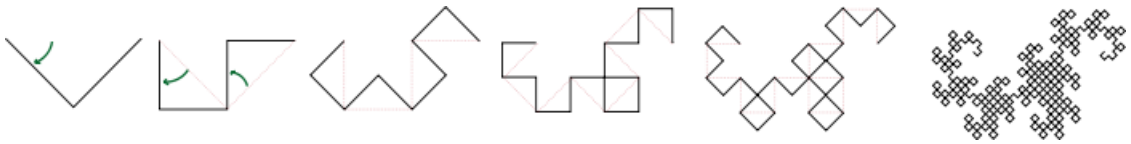

Figura 1: Exemplo

Este trajeto pode ser descrito através de um vetor de caracteres em que:

- `l` significa uma curva de 90 graus para a esquerda
- `r` significa uma curva de 90 graus para a direita
- `a` significa mover-se para frente
- `X` e `Y` correspondem a segmentos que podem ser dobrados

Você deve estar pensando: "ok, mas como vou desenhar isso na tela?". Este projeto irá explorar o uso de arquivos PostScript (`PS`) para tal visualização. **Considere o arquivo `dragonCurves.c` para implementação deste projeto**.

2 Especificação

As funções abaixo fazem uso da seguinte estrutura:

```
struct tab{
    char* mem; // vetor dinâmico
    int max; //tamanho do vetor
    int nb; //posições ocupadas do vetor
};
```

Sua tarefa neste projeto consiste em implementar (ou complementar a implementação) das seguintes funções:

1. `struct tab* criar_tab()`

   Aloca dinamicamente a estrutura, inicializa seus atributos. Considere que inicialmente ela não possui nenhum caracter guardado (`tab->nb=0`) e espaço para armazenar 16 caracteres (`tab->max=16` e vetor `t->mem` possui 16 posições alocadas dinamicamente).

2. `void destruir_tab(struct tab* t)`

   Libera o espaço de memória utilizado pelos atributos da estrutura bem como pela própria estrutura.

3. `void adicionar_char(struct tab* t, char c)`

   Observe que não é possível estabelecer a priori um tamanho fixo para o vetor que armazena or caracteres. Portanto, para implementar o fractal, é necessário definir um vetor dinâmico de tal forma que seja possível alterar seu tamanho sempre que necessário com uso da função `realloc`.

   Esta função coloca o caracter `c` na primeira posição livre do vetor da estrutura, o que requer a atualização dos atributos internos da estrutura. Caso não haja espaço suficiente no vetor, deve-se redimensionar o mesmo (`realloc`) pela seguinte regra:

   - Se nb $\leq$ 16, então max = 16;
   - Se 16 $<$ nb $\leq$ 32, então max = 32;
   - Se 32 $<$ nb $\leq$ 64, então max = 64;
   - ... e assim por diante.

   Seguindo tal regra, o crescimento do vetor é realizado de forma exponencial (fazendo que a quantidade de chamadas de `realloc` seja logaritmica).

4. `void adicionar_string(struct tab* t, char* s)`

   Concatena um vetor de caracteres com o vetor da estrutura de dados. Com isso, os atributos internos devem ser ajustados de acordo. Se for necessário redimensionar o vetor da estrutura (para que todo vetor `s` "caiba"), considere a regra implementada pela função `adicionar_char`.

5. `void imprimir_tab(struct tab* t)`

   Imprime o conteúdo do vetor da estrutura.

6. `void esvaziar_tab(struct tab* t)`

   Esvazia o vetor da estrutura, ou seja, `t->nb = 0` (o que é diferente de liberar a memória do mesmo).

7. `struct tab* subst(struct tab* arg)`

   Esta função aplica uma "rodada" da regra descrita abaixo, correspondende a um passo do fractal. Isso implica em gerar uma nova estrutura com o vetor atualizado. Em suma, dado um vetor de caracteres, o cálculo do fractal pode ser feito da seguinte forma:

   - Toda ocorrência de `X` no vetor é substituída por `XlYal`
   - Toda ocorrência de Y no vetor é substituída por `raXrY`
   - qualquer outro caracter é copiado sem alteração

   Exemplo:

   ```
   X
   XlYal
   XlYallraXrYal
   XlYallraXrYallraXlYalrraXrYal
   ```

Estes passos podem evidentemente ser repetidos indefinidas vezes.

8. `struct tab* calculo(int nb)`

   Esta função cria uma estrutura com o vetor `mem` contendo apenas o caracter `X` e aplica `nb` vezes a regra de atualização do fractal. Observe que é preciso liberar estruturas de dados antigas através das funções criadas anteriormente para evitar que a memória do computador seja ocupada inutilmente.

9. `void gerar_postscript(struct tab* t, char* arq_out)`

   Esta função já está implementada. Ela lê uma estrutura e transforma a cadeia de caracteres associada em um desenho PostScript.

Para 15 iterações, a saída esperada para o programa é ilustrada na figura abaixo.

Figura 2: Saída do programa para 15 iterações.