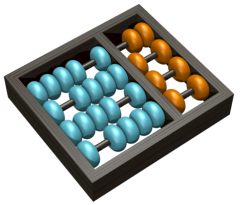
Instituto de Computação - UNICAMP

MC102 - Algoritmos e Programação de Computadores



Laboratório 08: *Superis Vincio Grandia*

Segundo Semestre de 2017 - Turmas Coordenadas

Peso da Atividade: 2

Prazo de Entrega: 29 de outubro de 2017 às 23:59:59

Conteúdo

Contexto

"E com cinco ou seis retas é fácil fazer um castelo."

– Toquinho, Aquarela

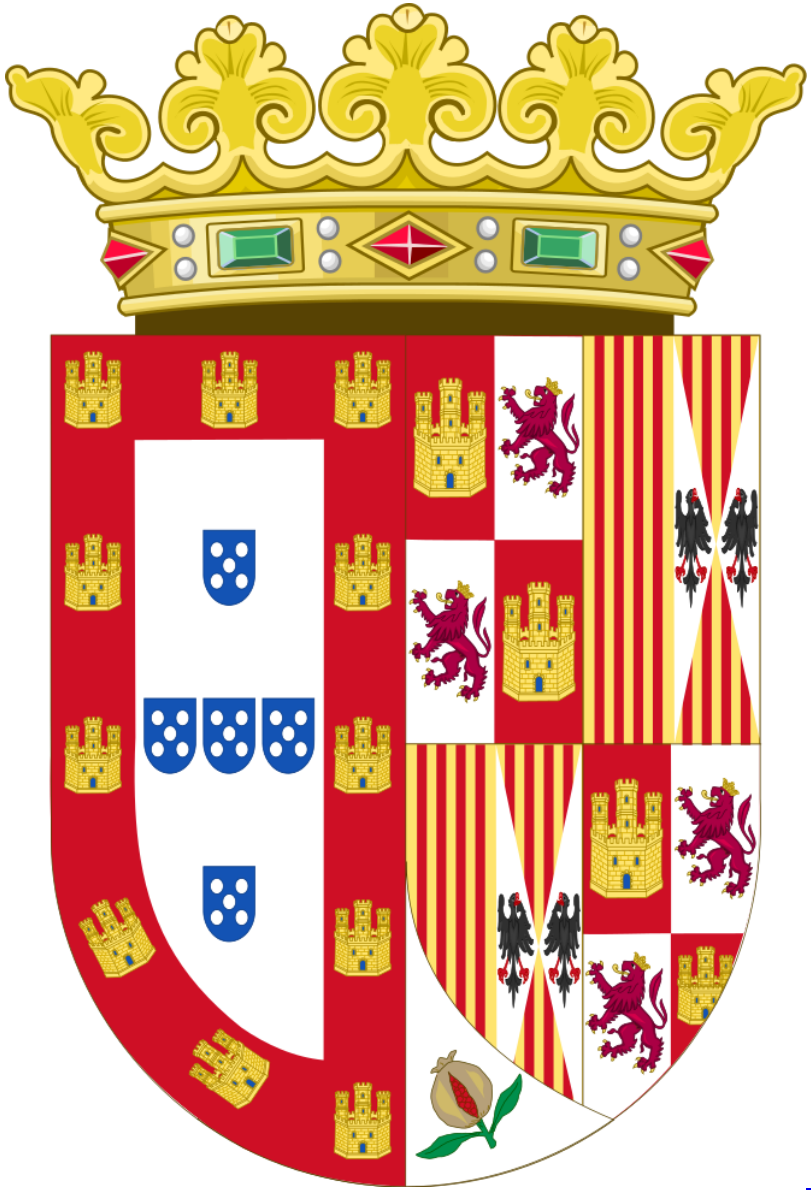


Figura 1: Escudo de Maria de Aragão. Extraído de Wikipedia.

A antiga dinastia de Pasárgada acaba de ser derrubada. O novo rei gostaria de alterar a bandeira, o brasão e o escudo, mas mantendo elementos com os quais o povo se relaciona, para manter a fidelidade de seus vassalos. Os principais símbolos nacionais contêm figuras que podem aparecer diversas vezes, com diversos tamanhos, em diversas posições, em diversas rotações e possivelmente refletidas.

Você faz parte da equipe de reestruturação heráldica real. Seus companheiros já reduziram as imagens dos principais símbolos nacionais a figuras em formato SVG, explicitando todas as operações que devem ser realizadas na imagem.

Tarefa

Nesta tarefa, você trabalhará com um formato real de arquivo de imagem: o formato SVG. Este formato é amplamente utilizado na web, publicações, comércio, arte, dentre muitas outras aplicações. Dois exemplos de imagens neste formato podem ser vistos nas Figuras 2 e 3.

Figura 2: Exemplo de imagem em SVG. Extraído de Wikimedia.

Figura 3: Exemplo de imagem em SVG. Extraído de openclipart.

O formato SVG não é um mapa de pontos como o formato PBM visto na tarefa 6. Ele é um formato de imagem vetorial, ou seja, ele descreve uma imagem por meio da composição de formas geométricas simples, como linhas, círculos, retângulos, polígonos, etc. Isto permite que uma imagem seja ampliada sem perda de resolução. (Experimente ampliar as imagens fornecidas para intuir sobre este fato.)

Nesta tarefa, a entrada e a saída são arquivos SVG simplificados, que contém polígonos a serem modificados. Esses arquivos podem ser abertos e visualizados em programas de imagem e navegadores, por exemplo.

Especificação informal do formato SVG

O começo de um arquivo SVG contém um cabeçalho com diversas meta-informações do arquivo. Nos arquivos tratados neste laboratório, este cabeçalho está contido no começo do arquivo, e pode ser descartado pelo seu programa.

Da mesma maneira que seus programas em C contém comentários delimitados por sequências de caracteres especiais, o SVG também pode conter comentários dentro de suas imagens, delimitados pela sequência `<!--` e pela sequência `-->`. Nesta tarefa, será utilizado um comentário na imagem de entrada para especificar diversas operações a serem aplicadas no restante da imagem original.

Para cada polígono na imagem, o SVG conterá uma linha na seguinte forma:

```
<polygon points='X1,Y1 X2,Y2 X3,Y3 [...] ' style= [...] />
```

Apenas a parte entre aspas simples é relevante para seu programa, pois conterá os pontos do polígono. Os 17 primeiros caracteres de cada linha:

```
<polygon points='
```

podem ser ignorados pelo seu programa.

Os caracteres depois da segunda aspas simples até o final da linha (`\n`):

```
style= [...] />
```

indicam informações adicionais do polígono, como cor e contorno. Seu programa não deve processar esta informação, ela deve ser simplesmente copiada para cada polígono resultante de uma transformação.

O final do arquivo SVG é marcado com a seguinte linha:

```
</svg>
```

Operações de transformação

A entrada do seu programa será uma imagem SVG que inclui, em seus comentários iniciais, uma sequência de transformações a serem aplicadas na imagem. A saída do seu programa será outra imagem SVG, com as transformações aplicadas a todos os polígonos da imagem original.

A sequência de transformações está codificada da seguinte maneira: cada operação é representada por uma letra maiúscula e, dependendo da operação, até dois parâmetros numéricos. O espaçamento entre as operações e seus parâmetros é desconsiderado. As operações e letras correspondentes são listadas a seguir:

- `C` indica que uma nova cópia da figura deve ser acrescentada na posição original para as transformações seguintes.
- `S x y` indica escala. O parâmetro `x` indica ampliação na horizontal e o `y` na vertical.

- `T x y` indica translação. O parâmetro `x` indica deslocamento à direita e o `y` para baixo.
- `R a` indica rotação. O argumento `a` é dado em graus.

1. Todas as operações são feitas usando o referencial global da imagem, ou seja, as transformações são aplicadas diretamente nas coordenadas `X` e `Y` de cada ponto do polígono.
2. Para a exibição das imagens, o SVG considera, por padrão, que o eixo horizontal cresce para a direita e o vertical para **baixo**.

Como consequência dos itens 1 e 2, a rotação, em particular, é realizada em torno da origem e o argumento `a` positivo rotaciona em sentido horário. Note que, ao aplicarmos diferentes operações em sequência, o resultado pode não parecer com nenhuma operação disponível. Ao aplicarmos rotação e escala, por exemplo, podemos obter um resultado semelhante a um cisalhamento. Todos os parâmetros das operações podem ser números negativos ou fracionários. Parâmetros negativos na operação de escala podem gerar reflexões ou até rotações na imagem. Após uma operação de `C` (de cópia), as transformações são aplicadas em uma nova cópia da imagem original. Cópias criadas anteriormente não serão mais alteradas.

Observe atentamente os exemplos na sua forma visual e de texto.

Código fornecido

Durante o desenvolvimento de programas, é comum utilizarmos funções desenvolvidas por terceiros sem o conhecimento preciso de seu funcionamento interno. Um exemplo disso, são as funções `scanf` e `printf`, que vêm sendo utilizadas na disciplina. Quando utilizamos uma função desta maneira, nos referimos a ela como uma "*caixa-preta*". Apesar de não ser necessário conhecer os detalhes de implementação de uma função caixa-preta, necessitamos saber algumas informações para sua utilização como: descrição dos parâmetros, valores de retorno, etc. Este conjunto de informações para se interagir com uma caixa-preta é denominado "*interface*".

Para a realização desta tarefa, serão fornecidas algumas funções caixa-preta para auxiliar na leitura do arquivo SVG. Estas funções devem ser utilizadas para realizar as operações de entrada e saída. Portanto, as funções definidas por você (incluindo a função `main`) não devem utilizar as funções `scanf` ou `printf`. Além das funções fornecidas, você deve definir outras funções, para dividir seu problema e implementar as operações organizadamente.

O código fornecido está contido no arquivo `lab08.h`. Dentro deste arquivo, estão definidas múltiplas funções auxiliares, mas apenas as três especificadas devem ser invocadas pelo código que você desenvolverá. A seguir, descrevemos a interface destas três funções:

1. `void scan_svg(int n[],double p[][2],char s[])`
Esta função será utilizada para ler as informações do SVG. A cada chamada, a função irá sobrescrever os três parâmetros da função com alguma informação do SVG.

Parâmetros de entrada da função

- `int n[]` : a primeira posição deste vetor recebe o número de operações da entrada ou o número de pontos de um polígono, ou `-1`, se estiver no fim do arquivo.
- `double p[][2]` : recebe os parâmetros das operações ou as coordenadas (x,y) dos pontos do polígono.
- `char s[]` : recebe os caracteres representando as operações ou a cadeia de formatação de um polígono.

Valor de retorno da função

- A função não retorna nenhum valor (a função é tipo `void`)

Na **primeira vez** que esta função é chamada, a sequência de operações de transformação (conforme descrito anteriormente) será lida. Resumidamente, para $0 \le i < n[0]$, a operação indicada por `s[i]` terá (possívelmente) parâmetros `p[i][0]` e `p[i][1]`. Note que seu programa não deve utilizar os valores `s[i]` e `p[i][_]` para $i \ge n[0]$.

Nas **vezes seguintes** que esta função é chamada, um polígono é lido (caso o arquivo não tenha chegado a seu fim): ou seja, para $0 \le i < n[0]$, as variáveis `p[i][0]` e `p[i][1]` contêm as coordenadas horizontal e vertical, respectivamente, do `i`-ésimo ponto do polígono. O vetor `s[]` não será processado, apenas retransmitido à função `print_svg` (descrita a seguir) para cada cópia do polígono. Não havendo mais polígonos a serem lidos, a função coloca `-1` em `n[0]`. Seu programa não deve utilizar os valores `p[i][0]` e `p[i][1]` para $i \ge n[0]$. A função também ignora partes da figura que não devem ser tratadas por seu programa, utilizadas como pano de fundo nos casos de teste.

2. **`void print_svg(int n[],double p[][2],char s[])`**
Esta função será utilizada para imprimir os polígonos transformados de volta ao formato SVG após terem sido processados pelo seu programa. A qualquer momento, esta função tem o mesmo comportamento e imprime o polígono passado como parâmetro.

Parâmetros de entrada da função

- `int n[]` : um vetor cuja primeira posição conterá o número de pontos do polígono que será impresso.
- `double p[][2]` : conterá o vetor com as coordenadas X e Y de cada ponto do polígono.
- `char s[]` : informações visuais do polígono; deve ser passado exatamente como preenchido por `scan_svg`.

Valor de retorno da função

- A função não retorna nenhum valor (a função é tipo `void`)

O número de chamadas deve ser exatamente igual ao número de polígonos na imagem final. Este número dependerá da quantidade de operações do tipo `C` e de polígonos na entrada.

3. **`void close_svg()`**
Esta função será utilizada para marcar o final do arquivo SVG. Ela não contém parâmetro algum e deve ser chamada apenas uma vez,

ao final de seu programa.

Sinta-se livre para inspecionar o código fornecido no arquivo `lab08.h`. Ao final desta disciplina, você terá visto todas as ferramentas necessárias para implementar códigos semelhantes.

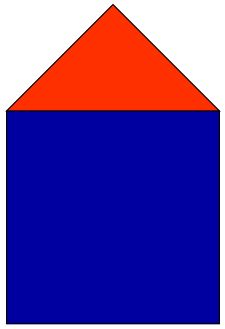
Também é fornecido o código `exemplo.c` que utiliza as três funções de `lab08.h` como caixa-preta. Você pode estudar este código e utilizá-lo opcionalmente como base para realizar sua tarefa.

Exemplos

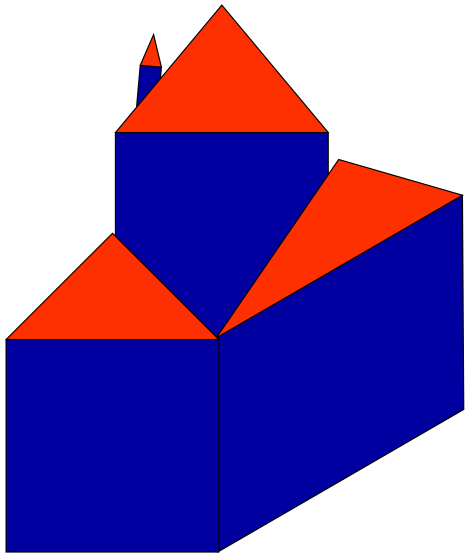
> *Notas:*
>
> Textos em `azul` designam dados de entrada, isto é, que devem ser lidos pelo seu programa.
> Textos em `preto` designam dados de saída, ou seja, que devem ser impressos pelo seu programa.

Exemplo de execução 1:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3
<svg  height='1024' width='1024' xmlns='http://www.w3.org/200
<!--
C
S 1.0 1.2
T -70 -290
C
R 45
S 2.0 0.9
R -66
S 1.0 0.65
T 47 366
C
T -256 150
C
S 0.1 0.3
R 5
T 281.6 -13.6
-->
<polygon points='330.981,437.019 693.019,437.019 693.019,799.
```



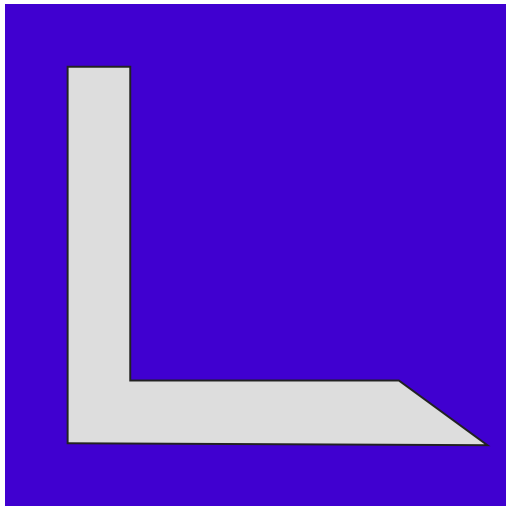
```
<polygon points='512,256 330.981,437.019 693.019,437.019 ' st
</svg>
```



```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3
<svg  height='1024' width='1024' xmlns='http://www.w3.org/200
<polygon points='260.981,234.423 623.019,234.423 623.019,668.
<polygon points='432.503,584.263 851.232,341.148 853.464,706.
<polygon points='74.981,587.019 437.019,587.019 437.019,949.0
<polygon points='303.146,119.891 339.212,123.047 329.745,231.
<polygon points='442.000,17.200 260.981,234.423 623.019,234.4
<polygon points='640.752,280.236 432.503,584.263 851.232,341.
<polygon points='256.000,406.000 74.981,587.019 437.019,587.0
<polygon points='325.912,67.370 303.146,119.891 339.212,123.0
</svg>
```

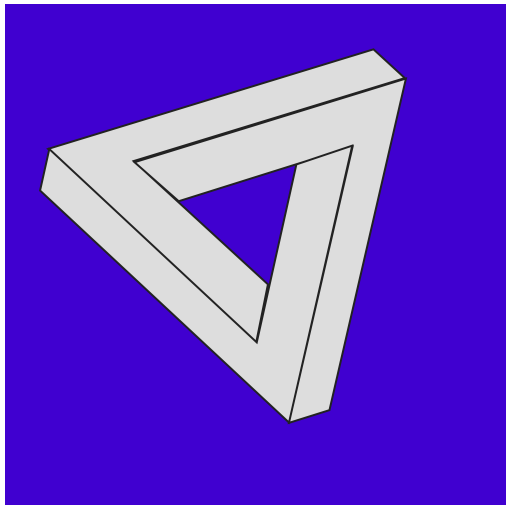
Exemplo de execução 2:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3
<svg height='1024' width='1024' xmlns='http://www.w3.org/2000
<rect height='1024' width='1024' style='fill:#4000D0;' />
<!--
C
R 60
S 1 0.53
T 800 0
C
R 60
S 1 0.53
R 120
T 720 917
C
R 60
S 1 0.53
R 240
T -32 387
-->
<polygon points='128,128 256,128 256,768 805,768 986,900 128,
```



```
</svg>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3
<svg height='1024' width='1024' xmlns='http://www.w3.org/2000
<rect height='1024' width='1024' style='fill:#4000D0;' />
<polygon points='753.149,92.671 817.149,151.422 262.892,321.0
<polygon points='663.170,830.090 580.290,856.140 710.540,291.
<polygon points='71.681,381.239 90.561,296.438 514.567,691.63
</svg>
```



No exemplo de execução 1, a primeira cópia da casa se torna a torre, a segunda se torna a parede lateral, seguida pela parede frontal e terminando com a chaminé.

No exemplo de execução 2, existe apenas um polígono hexalátero, que é copiado três vezes. Cada cópia é transformada e posicionada de forma a compor a imagem final. Note que esta imagem também contém elementos de pano de fundo antes das operações (no caso, apenas um elemento '`rect`angle') que é simplesmente copiado para a imagem final.

Nos outros casos de teste, pode haver um grande número de elementos usados como pano de fundo, além dos polígonos que serão transformados pelo programa.

Observações da Tarefa

- Cada polígono terá no máximo 500 pontos.
- O vetor de atributos visuais de cada polígono terá no máximo 256 caracteres.
- O número máximo de operações será 60000.
- Não existe limite para o número de polígonos na entrada: seu programa deve funcionar para um número arbitrário. (Dica: você não precisa armazenar todos os polígonos simultaneamente.)
- É garantido que o arquivo de entrada conterá no mínimo uma operação e um polígono.
- É garantido que, para cada operação `T`,`S` e `R`, existe pelo menos uma operação `C` que a antecede.
- É garantido que todo polígono conterá no mínimo um ponto.
- Utilize variáveis do tipo `double` nos cálculos matemáticos de coordenadas.
- As operações devem preservar a ordem dos polígonos. Ou seja, na saída, todos os polígonos mantêm sua ordem relativa e as cópias aparecem sempre juntas.

- Caso necessite utilizar a constante matemática π, a defina como: `3.14159265358979323846` .

Observações Gerais

- O número máximo de submissões é 20.
- O arquivo `lab08.c` deve conter todo o seu programa.
- Para a realização dos testes automáticos, a compilação se dará da seguinte forma: `gcc lab08.c -o lab08 -lm -Wall -Werror -ansi -pedantic` .
- Não se esqueça de incluir no início do programa uma breve descrição dos objetivos, da entrada, da saída, seu nome, RA e turma.
- Após cada submissão, você deve aguardar um minuto até poder submeter seu trabalho novamente.
- Ao completar esta tarefa você terá aprendido a utilizar e implementar funções.

Critérios Importantes

O **não** cumprimento dos critérios abaixo acarretará em **nota zero na atividade**, independentemente dos resultados dos testes do SuSy.

- Sua solução deve atender todos os requisitos definidos no enunciado.
- Não serão aceitas soluções contendo estruturas não vistas em sala (para este laboratório, poderão ser utilizadas apenas variáveis simples, vetores, matrizes, operações de entrada e saída, operações aritméticas, desvios condicionais, estruturas de repetição, strings e funções).
- Não é permitido o uso de `continue` e `break` (exceto em estruturas do tipo *switch-case*).
- Não é permitido o uso de variáveis globais.
- Cada função deve conter apenas um único `return`.
- Os únicos cabeçalhos aceitos para inclusão são `lab08.h` e `math.h`.
- Particularmente nesta tarefa, não é permitido a utilização das funções `scanf` e `printf`.