Trabalho I

Os alunos de Estruturas de Dados do Departamento de Ciência da Computação da UEL lançaram um jogo de estrondoso sucesso: a Bocha Geométrica!

O jogo é composto por formas espalhadas no solo e algumas delas espalhadas na arena de combate e por alguns disparadores. Cada disparador está posicionado em um certo ponto da arena e é responsável por impulsionar formas em uma certa direção.

Um disparador (Ilustração 1) possui 2 carregadores (esquerda e direita), 2 botões de seleção de carga e um botão de disparo.

Os carregadores são municiados com formas espalhadas no solo e são encaixados em disparadores. O botão de seleção de carga da direita (botão marrom) retira o elemento do topo da carga direita e coloca-o em posição de disparo. Se, por ventura, já houvesse alguma carga em posição de disparo, esta carga primeiro é colocada no topo da carga esquerda, liberando assim a posição de disparo. Similarmente, o botão de carga da esquerda (botão lilás), coloca o elemento do topo da carga esquerda em posição de disparo, desocupando previamente a posição de disparo.

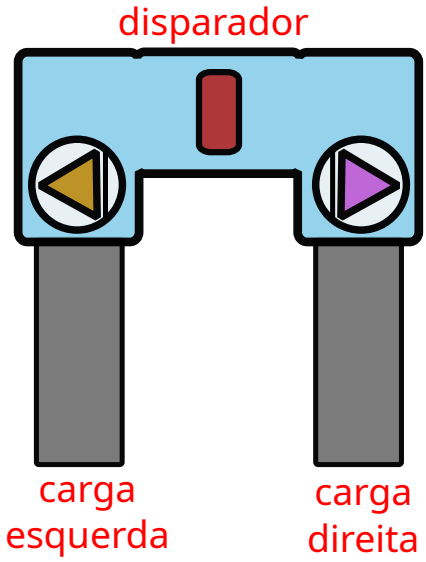

Ilustração 1: Disparador

Os disparos espalham formas na arena, porém, a ordem em que foram colocadas na arena é importante. Depois de um certo número de lançamentos, a formas são processadas na ordem que foram lançadas. Assim, considerando o i-ésimo elemento (I) lançado com (i+1)-ésimo elemento (J):

- Caso ocorra sobreposição entre estes elementos
  - elemento I tem área menor que elemento J: I é “esmagado”, portanto **destruído**. J volta para o chão
  - elemento I tem área maior que elemento J: I muda a cor da borda de J. Cor da borda de J passa a ser a cor de preenchimento de I. Ambos voltam para o chão (na mesma ordem relativa em que foram lançados). O elemento I é clonado, porém, intercambiando as cores de borda e preenchimento. O elemento clonado “volta” para o chão após os elementos I e J.
- Caso não ocorra sobreposição: ambos voltam para o chão, na mesma ordem relativa em que foram lançados.

A pontuação do jogo é a soma total das áreas esmagadas.

O objetivo do jogo é somar o maior número de pontos com o menor número de movimentos.

A Entrada

__ATENÇÃO__: Não esqueça de ler a descrição geral dos projetos (Sala de Aula).

A entrada do algoritmo será basicamente um conjunto de formas geométricas básicas (retângulos, círculos, etc) dispostos numa região do plano cartesiano .

Considere a Ilustração 2. Cada forma geométrica é definida por uma coordenada âncora (marcada, na figura, por um pequeno ponto vermelho) e por suas dimensões. A coordenada âncora do círculo é o seu centro e sua dimensão é definida por seu raio (r, na figura). A coordenada âncora do retângulo é seu canto inferior esquerdo[1] e suas dimensões são sua largura (w) e sua altura (h). A coordenada âncora de um texto, normalmente, é o início do texto, porém, pode ser definida como o meio ou o fim do texto. Por fim, uma linha é determinada por duas âncoras em suas extremidades. As coordenadas que posicionam as formas geométricas são valores reais.

Cada forma geométrica é identificada por um número inteiro.

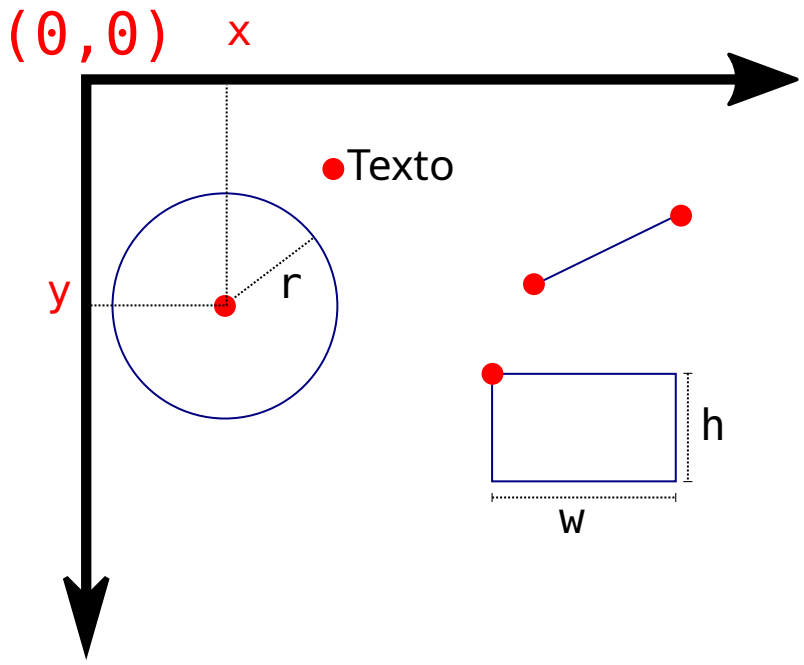

Ilustração 2: Formas no plano

As tabelas abaixo mostram os formatos dos arquivos de entrada (.geo e .qry). Cada comando tem um certo número de parâmetros. Os parâmetros mais comuns são:

- `i,j,k`: número inteiro, maior ou igual a 1. Identificador de uma forma geométrica.
- `r`: número real. Raio do círculo.
- `x,y`: números reais. Coordenada (x,y).
- `cor`: string. Cor válida dentro do padrão SVG.[2]

1 Note que o plano cartesiano está desenhado "de ponta-cabeça" em relação à representação usual.

2 http://www.december.com/html/spec/colorsvg.html.
https://www.w3.org/Graphics/SVG/IG/resources/svgprimer.html

comando	parâmetros	descrição
c	`i x y r corb corp`	*Cria um círculo com identificador i: (x,y) é o centro do círculo; r, seu raio,* corb *é a cor da borda e* corp *é a cor do preenchimento*
r	`i x y w h corb corp`	*Cria um retângulo com identificador i: (x,y) é a âncora do retângulo, w é a largura do retângulo e h, a altura.* corb *é a cor da borda e* corp *é a cor do preenchimento*
l	`i x1 y1 x2 y2 cor`	*Cria uma linha com identificador i, com extremidades nos pontos (x1,y1) e (x2,y2), com a cor especificada.*
t	`i x y corb corp a txto`	*Cria o texto* txto *com identificador i, nas coordenadas (***x,y***) e com a cores indicadas.* corb *é a cor da borda e* corp *é a cor do preenchimento. O parâmetro* **a** *determina a posição da âncora do texto:* **i**, *no início;* **m**, *no meio,* **f**, *no fim.* *O texto txto é o último parâmetro do comando. Pode incluir espaços em branco e se estende até o final da linha.*
ts	`fFamily fWeight fSize`	*Muda o estilo dos textos (comando t) subsequentes.* *font family: sans (sans-serif), serif, cursive;* *font weight (n: normal, b: bold, b+: bolder, l: | lighter)*
comandos .geo		

Cada tipo de figura está espalhada no chão, mas sabe-se a ordem em que foram criadas. Figuras são retiradas do chão (na ordem em que foram colocadas lá) e são municiadas em carregadores. Carregadores são encaixados em disparadores. Disparadores disparam formas na arena. Formas da arena podem voltar para o chão.

O programa a ser executado está no arquivo .qry que contém comandos do processador.

comando	parâmetros	descrição
pd	`l x y`	*Posiciona o disparador `l` na coordenada `(x,y)`*
lc	`c n`	*Coloca no carregador c as primeiras n formas que estão no chão.* *TXT: reportar os dados de cada uma das figuras carregadas*
atch	`d cesq cdir`	*Encaixa no disparador d os carregadores cesq (na esquerda) e cdir (na direita)*
shft	`d [e|d] n`	*Pressiona o botão esquerdo (e) ou o botão direito (d) do disparador d n vezes* *TXT. Reportar os dados da figura que resultou ficar no ponto de disparo*
dsp	`d dx dy [v|i]`	*Posiciona a forma que está em posição de disparo a um descolamento de dx, dy em relação à posição do disparador.* *TXT: reportar os dados da forma disparada e a posição final da forma* *SVG: se o último campo for v, anotar as dimensões do disparo, como mostrado na Ilustração 3.*
rjd	`d [e|d] dx dy ix iy`	*Rajada de disparos. Equivalente a uma sequência de disparos na forma:* **`shft d [e|d] 1`** **`dsp d dx+i*ix dy+i*iy`** *Até as formas do respectivo carregador se esgotarem. Os valores ix e iy são valores a serem acrescentados aos deslocamentos dx e dy (respectivamente) a cada disparo.* *TXT: reportar dados das forma disparadas*
calc		*Processa as figuras da arena conforme descrito anteriormente.* *TXT.: reportar o resultado de cada verificação; a área total esmagada no round e a área total esmagada* *SVG: colocar um asterisco vermelho no local da forma esmagada.*
Comandos .qry		

A seguir, é mostrado um exemplo de um arquivo de consulta (a.qry). Este arquivo posiciona dois disparadores, carrega 5 carregadores com as formas especificadas num arquivo .geo (não mostrado). A seguir, encaixa dois carregadores no disparador 1 e outros dois carregadores no disparador 2. Na sequência, rotaciona 4 formas para esquerda no disparador 2 e 1 forma para direita

no disparador 1. Finalmente, é efetuado o disparo no disparador 1 e outro disparo no disparador 2. A Ilustração 3 mostra a posição final das formas disparadas, supondo que ambas as formas fossem retângulos.

```
pd 1 0.0 0.0
pd 2 0.0 100.0
lc 6 20
lc 7 25
lc 9 15
lc 3 25
lc 11 35
atch 1 6 7
atch 2 3 11
shft 2 e 4
shft 1 d 1
dsp 1 20.0 27.35
dsp 2 14.95 -70.0
calc
```

Exemplo: a.qry

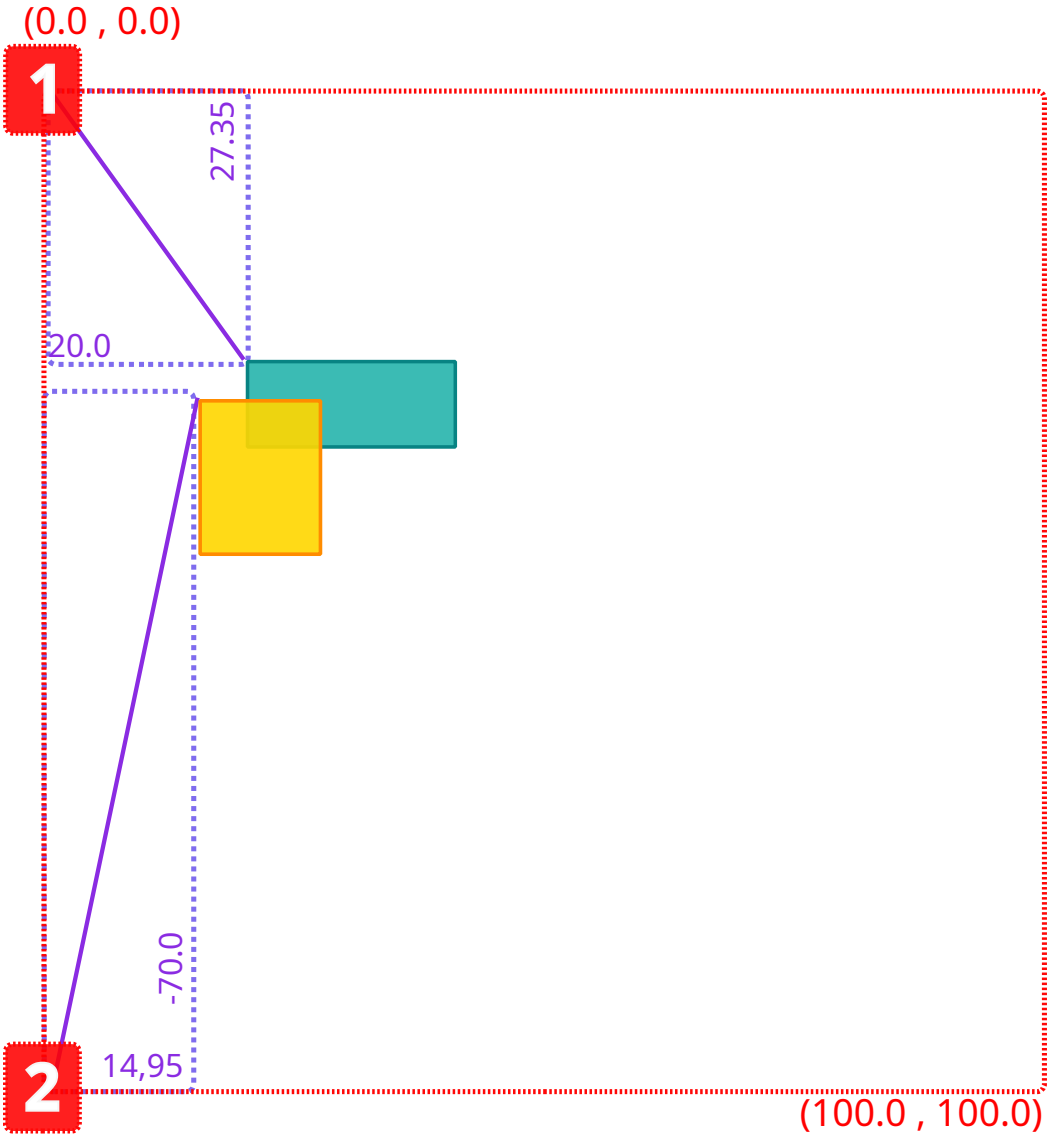

Ilustração 3: Exemplo parcial de uma consulta (a.qry)

Sobreposição

Os efeitos dos disparos dependem da ocorrência (ou não) de sobreposições e das áreas dos elementos i e i+1.

A sobreposição se dá quando as formas possuem alguma intersecção. A determinação da interseção entre as formas geométricas é trivial. Para determinar a ocorrência de interseção de um texto, fazemos a seguinte convenção:

- Seja: (x_t,y_t) a âncora do texto, $|t|$ o número de caracteres do texto
- O texto é considerado como se fosse um segmento (x_1,y_1) - (x_2,y_2). Este segmento é usado para determinar a ocorrência de colisões (veja Ilustração 4).
- se posição da âncora do texto for:
  - **'i'**: `x1 = xt, y1 = yt,`
    `x2 = xt + 10.0 * |t|, y2 = yt`
  - **'f':** `x1 = xt - 10.0 * |t| , y1 = yt,`
    `x2 = xt, y2 = yt`
  - **'m'**: `x1 = xt - 10.0 * |t| / 2 , y1 = yt,`

```
x2 = xt + 10.0 * |t| / 2 ,  y2 = yt
```

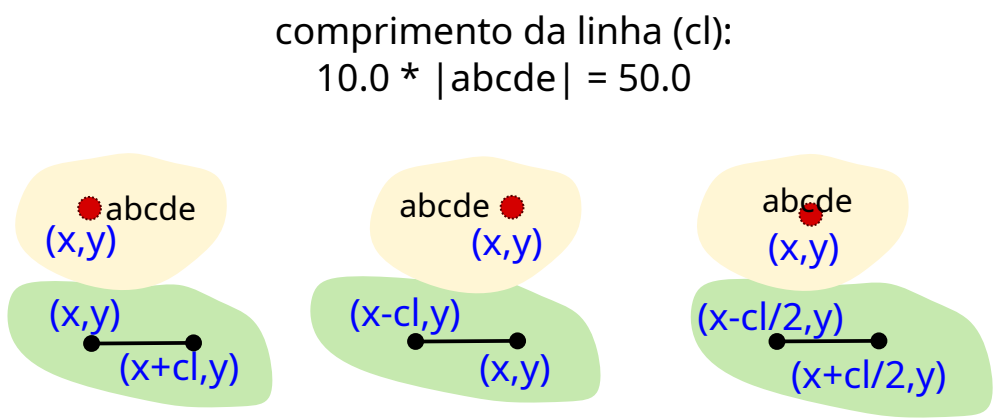

Ilustração 4: Segmento induzido pelo texto

Para o cálculo da área de textos e linhas considerar a seguinte convenção:

- área do texto: `20.0 * número de caracteres`
- área da linha: `2.0  * comprimento da linha`

Clonagem

A clonagem consiste em criar outra forma semelhante à original, ou seja, com os mesmos valores de atributos, exceto o código que deve ser um um valor único.[3]

A Saída

A saída textual é um arquivo texto comum contendo as informações solicitadas pelas instruções contidas no arquivo .qry.

A saída gráfica é um arquivo .svg que, renderizado por algum aplicativo, mostre pictoricamente as formas espalhadas no chão.

Ao final do processamento do arquivo .geo, deve ser produzido um arquivo .svg mostrando todas as formas existentes no chão. Da mesma forma, ao final do processamento do arquivo .qry, outro arquivo .svg deve ser produzido mostrando as formas remanescente no chão, além das "anotações" indicadas na descrição dos comandos. Note que instruções do .qry podem remover formas (que não aparecerão no .svg), modificar atributos de algumas formas (apenas os atributos finais aparecerão no .svg) ou criar novas formas (que aparecerão no arquivo .svg).

No final do processamento do arquivo qry, devem ser acrescentadas ao arquivo .txt as seguintes informações:

- pontuação final
- número total de instruções executadas
- número total de disparos
- número total de formas esmagadas
- número total de formas clonadas

3 Dica: manter o maior código, incrementar este código e atribuir ao clone.

IMPLEMENTAÇÃO

Implementar dos TADs postados em nossa Sala de Aula.
É **terminantemente proibido** declarar structs nos arquivos de cabeçalho (.h).
O programa deve estar bem modularizado (arquivos .h e .c). Cada estrutura de dados deve estar em um módulo separado. O arquivo .h deve estar muito bem documentado (lembre-se que é um "contrato").

AVALIAÇÃO

Espera-se uma atitude pró-ativa para a aquisição dos conhecimentos (i.e., estudo) para resolver o problema proposto.
A avaliação consistirá da execução dos testes e da inspeção de código. A nota será proporcional ao número de testes bem sucedidos, aplicados os descontos escritos abaixo.
ATENÇÃO: Os focos do projeto são: implementação e uso das estrutura de dados Pilha e Fila, modularização bem feita. Hipoteticamente, se todo o resto estiver bem feito e esta parte mal feita, a nota será muito baixa.

ATENÇÃO: Caso algum tipo de FRAUDE seja detectada: Nota ZERO a TODOS os envolvidos.

Critério	Desconto
Escrever struct em arquivo .h	2.5
Modularização Pobre: .h mal projetado, mal documentado	até 2.0
Não implementado conforme especificado	tipicamente até 4.0, mas em casos graves, o desconto pode ser total
• não implementar estruturas pedidas (gravíssimo)	
Procedimentos extensos e/ou complicados	até 1.0
Escolha/uso de estrutura pouco eficiente	até 2.0
Poucos commits ou commits concentrados perto do final do prazo.	tipicamente até 3.0; porém, o padrão de commit pode indicar possível fraude.
Implementação ineficiente e/ou desidiosa.	Até 1.5.
Não usar o makefile provido	Em geral, nenhum desconto. Mas, se ocorrer problema de compilação ou execução que poderia ter sido evitado pelo uso do modelo do makefile provido, a consequência pode ser grave.
Erro de compilação	Nenhum teste será executado. Portanto, nota Zero, especialmente se for causado por negligência do aluno.

O Que Entregar

Os arquivos fontes devem estar em um repositório GIT. Submeter ao Classroom um arquivo-texto (.txt) com uma linha, com o formato abaixo. A URL fornecida será usada para clonar o repositório (git clone)

```
apelido url-do-repositorio
```

Exemplo:

```
joseMane https://github.com/zemane/t2.git
```

O diretório clonado deve estar organizado como explicado na descrição geral.

RESUMO DOS PARÂMETROS DO PROGRAMA TED

Parâmetro / argumento	Opcional	Descrição
`-e` ***`path`***	S	Diretório-base de entrada (**BED**)
`-f` ***`arq`*`.geo`**	N	Arquivo com a descrição da cidade. Este arquivo deve estar sob o diretório **BED**.
`-o` ***`path`***	N	Diretório-base de saída (**BSD**)
`-q` ***`arqcons`*`.qry`**	S	Arquivo com consultas. Este arquivo deve estar sob o diretório **BED**.

RESUMO DOS ARQUIVOS PRODUZIDOS

-f	-q	comando com sufixo	arquivos
arq.geo			arq.svg
arq.geo	***arqcons***.qry		arq.svg arq-arqcons.svg arq-arqcons.txt
arq.geo	***arqcons***.qry	***sufx***	arq.svg arq-arqcons.svg arq-arqcons.txt arq-arqcons-sufx.[svg|txt][4]

ATENÇÃO:
* os fontes devem ser compilados com a opção `-fstack-protector-all`.
* adotamos o padrão C99. Usar a opção `-std=c99`.

4 Podem ser produzidos os respectivos arquivos .svg e/ou .txt, dependendo da especificação do comando.