ORGANIZAÇÃO DOS PROJETOS

Serão desenvolvidos alguns projetos durante o semestre. Todos serão organizados como descrito a seguir.

Basicamente, os programas lerão 2 arquivos: um arquivo que descreve um conjunto de dados a serem armazenados em alguma estrutura de dados e um arquivo descrevendo operações sobre o primeiro conjunto. As operações podem causar a delição, modificação, inserção de dados ao primeiro conjunto.
O resultado final do processamento de cada conjunto é gravado em um ou mais arquivos de saída.

ENTRADA DE DADOS

A entrada de dados, via de regra, ocorrerá por meio de um ou mais arquivos. Estes arquivos estarão sob um diretório, referenciado por **BED** neste texto.[1]

SAIDA DE DADOS

O dados produzidos serão mostrados na saída padrão e/ou em diversos arquivos-texto. Alguns resultados serão gráficos no formato SVG. Os arquivos de saída serão colocados sob um diretório, referenciado por **BSD** neste texto.[2]

ORGANIZAÇÃO DA ENTREGA

O trabalho deve ser submetido no formato **ZIP**, cujo nome deve ser curto, mas suficiente para identificar o aluno ou a equipe.[3] Este arquivo deve estar organizado como descrito à frente.

PROCESSO DE COMPILAÇÃO E TESTES DO TRABALHO

Organização do ZIP a ser entregue

A organização do zip a ser entregue pelo aluno deve ser a seguinte:

[abreviatura-nome]	*Por exemplo, josers.*
LEIA-ME.txt	*colocar matrícula e o nome do aluno. Atenção: O número da matricula de estar no início da primeira linha do arquivo. Só colocar os números; não colocar qualquer pontuação.*
*	*Outros arquivos podem ser solicitados a cada fase.*
/src	*(arquivos-fonte)*
makefile	*deve ter target para a geração do arquivo objeto de cada módulo e o target* **siguel** *que produzirá o executável de mesmo nome dentro do mesmo diretório* **src**. *Os fontes devem ser compilados com a opção -fstack-protector-all.* ** adotamos o padrão C99. Usar a opção -std=c99.*

1 Indicado pela opção -e.
2 Indicado pela opção -o.
3 Por exemplo, josers.zip (se aluno se chamar José Roberto da Silva), josers-mariabc.zip (para uma equipe com dois alunos. Evite usar maiúsculas, caracteres acentuados ou especiais.

`*.h e *.c`	***Atenção:*** *não devem existir outros arquivos além dos arquivos fontes e do makefile*

Organização do diretório para a compilação e correção dos trabalhos (no computador do professor):

[HOME_DIR]

`*.py`	*scripts para compilar e executar*
`\t`	*diretório contendo os arquivos de testes*
`*.geo *.qry`	*arquivos de consultas, talvez, distribuídos em alguns outros sub-diretórios*
`\alunos`	*(contém um diretório para cada aluno)*
`\abrnome`	*diretório pela expansão do arquivo submetido (p.e., josers)*
	outros subdiretórios para os arquivos de saída informados na opção `-o`

Os passos para correção serão os seguintes:

1. O arquivo .zip será descomprimido dentro do diretório alunos, conforme mostrado acima
2. O makefile provido pelo aluno será usado para compilar os módulos e produzir o executável. Os fontes serão compilados com o compilador gcc em um máquina virtual Linux. Os executáveis devem ser produzidos no mesmo diretório dos arquivos fontes O professor usará o GNU Make. Serão executadas (a partir dos scripts) o seguinte comando:
   - **`make t1`**
3. O programa será executado automaticamente várias vezes: uma vez para cada arquivo de testes e o resultado produzido será inspecionado visualmente pelo professor. Cada execução produzirá (pelo menos) um arquivo **.svg** diferente dentro do diretório informado na opção **-o**. Possivelmente serão produzidos outros arquivos .svg e .txt.

APENDICE

https://www.gnu.org/software/make/manual/make.html
http://opensourceforu.com/2012/06/gnu-make-in-detail-for-beginners/

Trabalho I

A Entrada

A entrada do algoritmo será basicamente um conjunto de retângulos e círculos dispostos numa região do plano cartesiano e, possivelmente, algumas consultas, por exemplo, que indagam se duas das formas geométricas se sobrepõem. Os comandos estão contidos num arquivo .geo e as consultas num arquivo .qry.

Considere a Ilustração 1. Cada forma geométrica é definida por uma coordenada âncora e por suas dimensões. A coordenada âncora do círculo é o seu centro e sua dimensão é definida por seu raio. A coordenada âncora do retângulo é seu canto inferior esquerdo[4] e suas dimensões são sua largura e sua altura. As coordenadas que posicionam as formas geométricas são valores reais. Cada forma geométrica é identificada por um número inteiro.

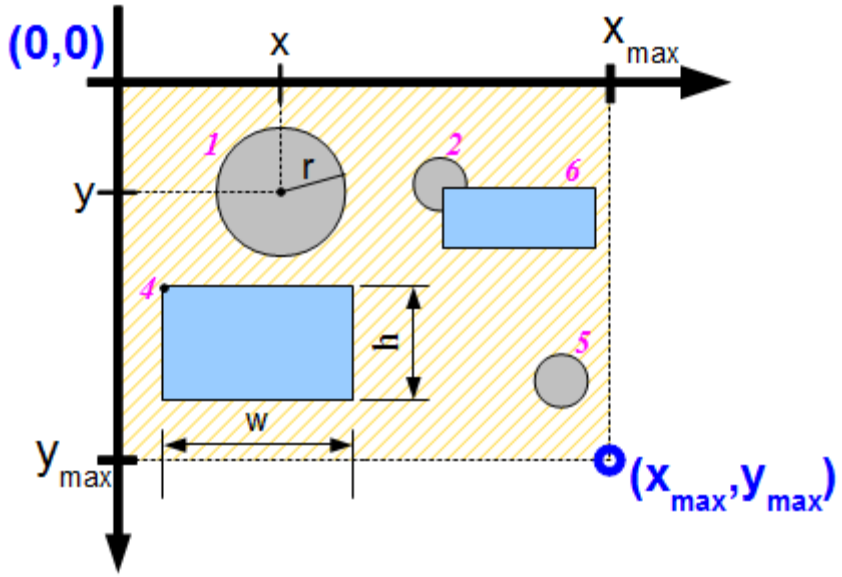

Ilustração 1

As tabelas abaixo mostram os formatos dos arquivos de entrada (.geo e .qry). Os arquivos de entrada são compostos, basicamente, por conjunto de comandos (um por linha), a saber: **`c`** (desenhe um círculo), **`r`** (desenhe um retâgulo), **`t`** (escreva um texto), etc.

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 criada pelos comandos **`c`** ou **`r`**.
- `r`: número real. Raio do círculo.
- `x,y`: números reais. Coordenada (x,y).
- `cor`: string. Cor válida dentro do padrão SVG.[5]

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

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

Alguns comandos utilizam memória auxiliar para armazenar identificadores

comando	parâmetros	descrição
c	`i r x y corb corp`	*desenhar círculo.* corb *é a cor da borda e* corp *é a cor do preenchimento*
r	`i w h x y corb corp`	*desenhar 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`	*Desenhar linha com extremidades nos pontos (x1,y1) e (x2,y2), com a cor especifiicada.*
t	`i x y corb corp txto`	*desenha o texto* txto *nas coordenadas (x,y) e com a cores indicadas.* corb *é a cor da borda e* corp *é a cor do preenchimento*
comandos .geo		

Algumas consultas fazem referência a pilhas, filas e listas. O programa deve manter 10 pilhas, 10 filas e 10 listas, identificadas por números de 0 à 9. Consultas sobre listas também podem fazer referência a registradores numerados de 0 a 9.

comando	parâmetros	descrição
o?	`j k id cor1 cor2`	*As formas geométricas cujos identificadores são* **j** *e* **k** *se sobrepõem?*[6] *(j e k não se referem a um texto)* *Acrescenta ao conjunto de linhas uma linha, com identificador id, entre as coordenadas âncoras das figuras identificadas por j e k. Se as formas se sobrepõem, muda suas cores de preenchimento para cor1; caso contrário, muda para cor2.* ***Saida*: *.txt:*** *copiar o comando e, na linha seguinte escrever uma mensagem informando o tipo das formas j e k e se sobrepõe ou não* ***.svg**: traçar uma linha conforme descrito.*

6 A borda da figura pertence à figura. Assim, as figuras que coincidem apenas nas bordas também se sobrepõem.

comando	parâmetros	descrição
i?	`j x y`	*O ponto (x,y) é interno à j-ésima forma geométrica?*[7] ***Saída**: .**txt**: copiar o texto da consulta e, na linha seguinte, informar o tipo da forma geométrica e se é interno ou não.* *.**svg**: colocar um ponto (pequeno círculo) nas coordenadas (x,y) e pintá-lo de azul se for interno e magenta se for externo. Colocar uma linha (da mesma cor do ponto) ligando o ponto ao centro de massa da figura j.*
pnt	`j corb corp`	*Mudar as cores da borda (*corb*) e do preenchimento (*corp*) da forma/texto de identificador* j. ***Saída**: .**txt**: copiar o texto da consulta e, na linha seguinte, informar as coordenadas originais da forma/texto.* *.**svg**: a figura afetada é mostrada com as novas cores*
delf	`j`	*Remove a forma/texto de identificador j.* ***Saida: .txt:** copiar o texto da consulta e mostrar todas as informações sobre esta forma/texto.* ***.svg:** o elemento removido não deve aparecer no svg.*
psh	`p i`	*Empilha na pilha p a forma/texto de identificador i*
pop	`p i dx dy prop`	*Remove elemento do topo da pilha p. Cria uma nova forma/texto, com identificador i, com cores de borda e preenchimento invertidas, transladados em x e y por dx e dy e com tamanho modificado segundo a proporção prop (ex, 0.8, significa reduzir a 80%; 1.0, manter o mesmo tamanho; 1.2, ampliar em 20%)* *txt: reportar elemento removido da pilha.*
inf	`f i`	*Semelhante psh. Insere na fila f.*
rmf	`f i dx dy prop`	*Semelhante pop. Retira da fila f.*
ii	`l i r`	*Insere a forma/texto de identificador i no inicio da lista l e armazena no registrador r o "atalho" para o elemento inserido*
if	`l i r`	*Semelhante ii. Insere no fim da lista*

7 Um ponto na borda da figura pertence à figura, **mas não é interno** à figura.

comando	parâmetros	descrição
ia	`l i ro rd`	*Insere o a forma/texto de identificador i na lista l antes do elemento cujo atalho está armazenado em ro e guarda o atalho do elemento recém-inserido em rd*
id	`l i ro rd`	*Semelhante ia, mas insere depois da forma/texto identificado por i.*
da	`l r`	*Remove de l o elemento anterior ao elemento cujo atalho está armazenado no registrador r.*
dd	`l r`	*Semelhante a da, mas remove depois da forma cujo atalho está armazenado em r.*
dpl	`l i dx dy prop incprop`	*Percorre a lista l, na sequência, do início ao fim. Cria novas formas, semelhantes às visitadas, porém, transladadas de dx e dy, cujos identificadores começam em i e são incrementados sequencialmente e seus tamanhos são modificados a partir da proporção prop. A cada nova forma a proporção é incrementada em incprop. Por exemplo, se prop é 0.8 e incprop é 0.1, a primeira nova forma será reduzida a 80% do tamanho original, a segunda será reduzida a 90%, a terceira manterá seu tamanho original, a quarta será aumentada em 10%, e assim sucessivamente. A lista l é esvaziada, mas as formas/textos não são apagados.*
Comandos .qry		

A figura abaixo mostra um exemplo de um arquivo de entrada (consistente com a Ilustração 1). Note que a extensão do arquivo é **.geo**. As primeiras operações desenham círculos e retângulos.

```
c 1 50.00 50.0 30.00 grey magenta
r 6 121.0 46.0 100.0 30.0 cyan yellow
c 2 grey magenta 120.0 45.0 15.0
r 4 10.0 150.0 90.0 40.0 cyan yellow
c 5 230.0 180.0 13.0 grey magenta
```

***a01*.geo**

```
o? 2 6
i? 5 210.0 160.0
```

q.qry

A Saída

O programa deverá produzir um arquivo **`.svg`** e um arquivo **`.txt`** ambos com o mesmo nome base do arquivo **`.geo`**.

O arquivo .svg produzido deve mostrar as formas geométricas. Além disso, para o comando **`o`**, caso as figuras identificadas se sobreponham, elas devem ser envolvidas por um retâgulo tracejado contendo a palavra "sobrepoe". Existem várias ferramentas que renderizam arquivos .svg. As figuras abaixo mostram um exemplo de arquivo **`.svg`** e sua respectiva renderização.

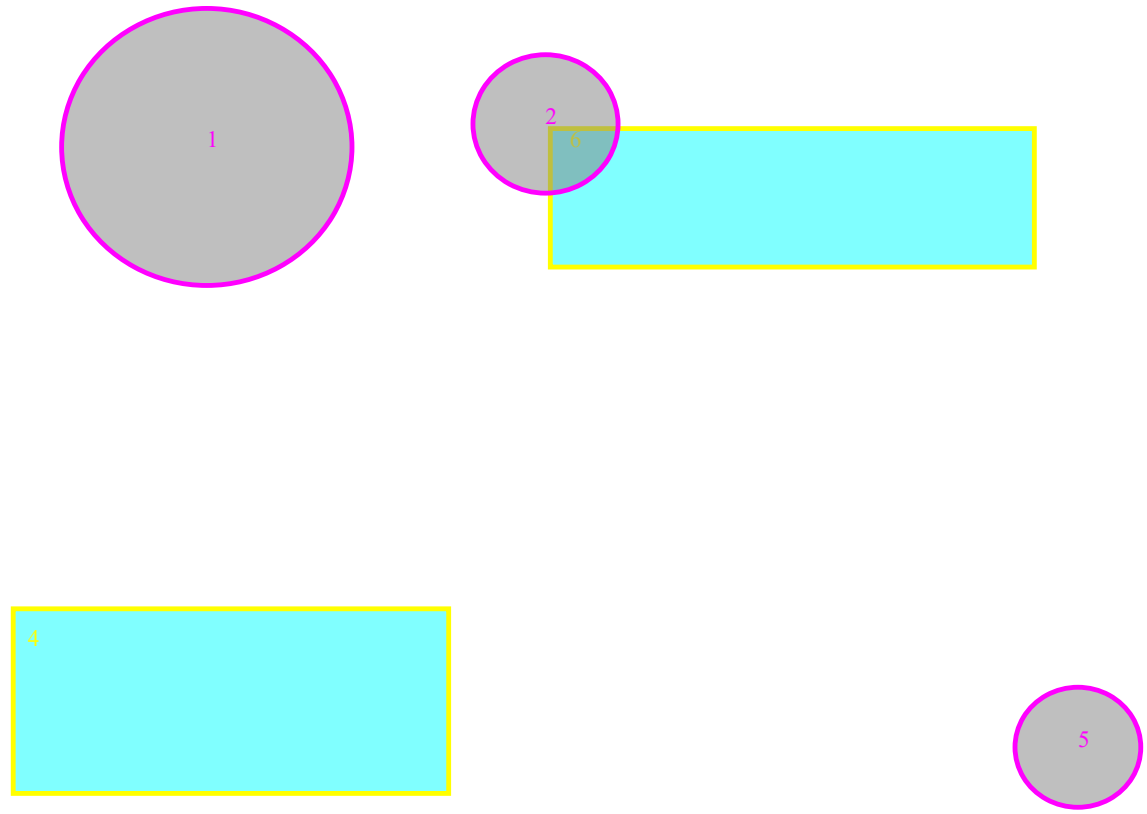

Illustration 2: Arquivo a01.svg

Se o arquivo .geo (a01.geo, no nosso exemplo) for processado em conjunto com um arquivo .qry (q.qry, no exemplo), além de a01.svg, deverá ser produzido o arquivo a01-q.svg, contendo os círculo, retângulos, etc acrescidos aos resultados da consulta.

Também produzir um arquivo-texto (a01-q.txt, no exemplo) contendo o resultado textual de todas as consultas. Neste arquivo deve ser copiado em uma linha o texto da consulta e, na linha seguinte, o seu resultado.

```
o? 2 6
2: retângulo 6: círculo SIM

i? 5 210.0 160.0
5: círculo NAO INTERNO
```

Arquivo a01-q.txt

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. Além disso, deverá ser produzido um vídeo de, aproximadamente, 5 minutos no qual apresenta o sistema. O aluno deve ter por objetivo convencer o avaliador que: (a) o programa funciona; (b)
o programa foi bem implementado O vídeo deve ser colocado no Youtube e o seu link deve estar anotado no arquivo leiame.

O PROGRAMA

O nome do programa deve ser **siguel** e aceitar quatro parâmetros:[8]

```
t1 [-e path] -f arq.geo [-q consulta.qry] -o dir
```

O primeiro parâmetro (**-e**) indica o diretório base de entrada. É opcional. Caso não seja informado, o diretório de entrada é o diretório corrente da aplicação. O segundo parâmetro (**-f**) especifica o nome do arquivo de entrada que deve ser encontrado sob o diretório informado pelo primeiro parâmetro. O terceiro parâmetro (-q) é um arquivo de consultas. O último parâmetro (**-o**) indica o diretório onde os arquivos de saída (***.svg e *.txt**) deve ser colocados. Note que o nome do arquivo pode ser precedido por um caminho relativo; ***dir e path*** é um caminho absoluto ou relativo (ao diretório corrente).

A seguir, alguns exemplos de possíveis invocações de **siguel**:

- **t1** -e /home/ed/testes/ -f t001.geo -o /home/ed/alunos/aluno1/o/
- **t1** -e /home/ed -f ts/t001.geo -o /home/ed/alunos/al1/o
- **t1** -f ./tsts/t001.geo -e /home/ed -o /home/ed/alunos/aluno1/o/
- **t1** -o ./alunos/aluno1/o -f ./testes/t001.geo
- **t1** -o ./alunos/aluno1/o -f ./testes/t001.geo -q ./t001/q1.qry
- **t1** -e ./testes -f t001.geo -o ./alunos/aluno1/o/ -q ./q1.qry

O Que Entregar

Submeter no Classroom o arquivo .zip com os fontes , conforme descrito anteriormente.

8 Novos parâmetros são acrescentados no decorrer do ano

RESUMO DOS PARÂMETROS DO PROGRAMA SIGUEL

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][9]

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

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