Universidade Federal de Minas Gerais **Algoritmos e Estruturas de Dados II**

Instituto de Ciências Exatas **2º Semestre de 2013**

Departamento de Ciência da Computação

Trabalho Prático 0 – Revisão de Programação C e Tipos Abstratos de Dados

Valor: 5 pontos

Data de entrega no PRATICO (aeds.dcc.ufmg.br): 02/09/2013

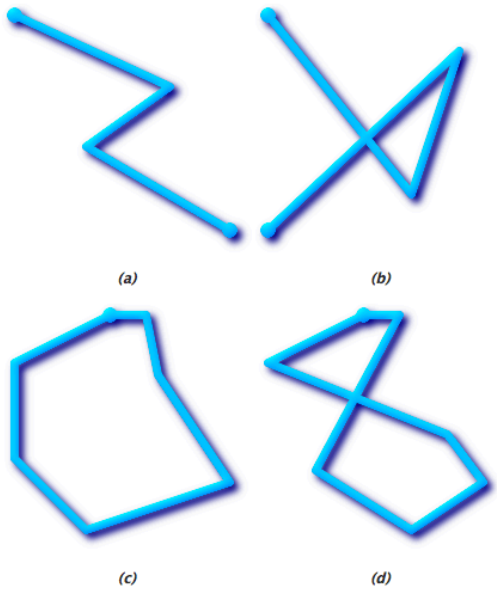
O objetivo deste trabalho é rever conceitos básicos de programação, especialmente em linguagem C, e explorar conceitos de Tipos Abstratos de Dados (TADs) e análise de complexidade.

Você deverá implementar estruturas para os seguintes tipos de dados em um TAD:

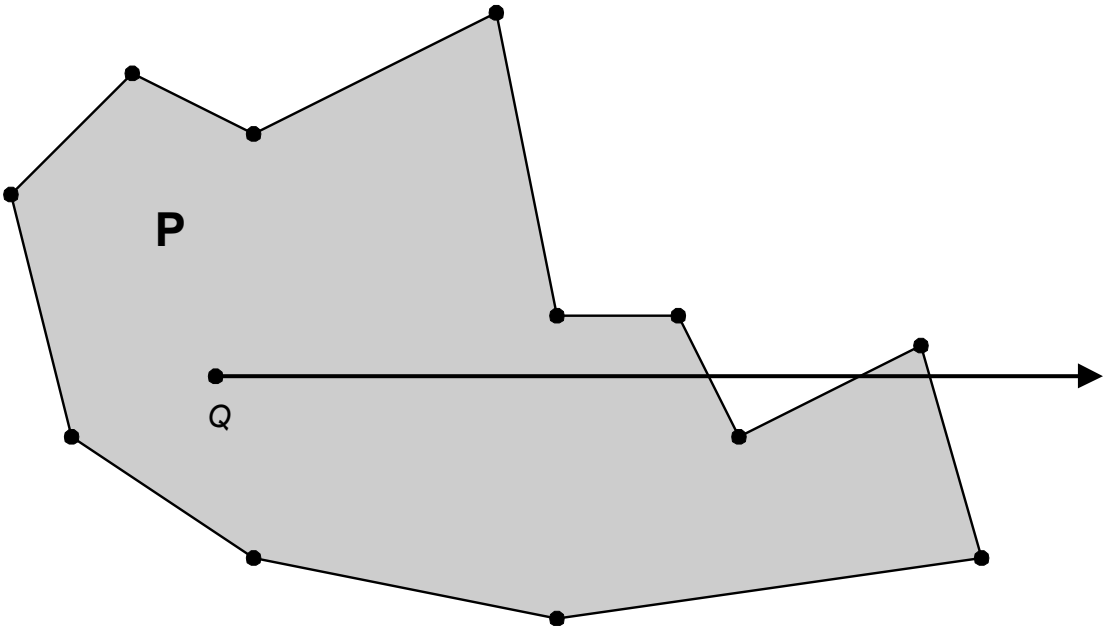
- **Ponto**: representado por coordenadas cartesianas x e y
- **Linha poligonal**: representada por uma sequência de 2 a 100 **pontos** (vértices ao longo da linha)
- **Polígono**: representado por uma sequência de 3 a 100 **pontos** (vértices ao longo do polígono), sendo que o último vértice e o primeiro coincidem, ou seja, têm coordenadas idênticas.

As coordenadas serão representadas por números de ponto flutuante (double). **Considerar que o número máximo de pontos, linhas e polígonos tratados pelo programa é 100 (cada).**

Dentre as funções que devem ser implementadas no TAD, três realizam processamento geométrico. Duas destinam-se a verificar se a linha poligonal ou o polígono têm auto-interseções, ou seja, são **não-simples**. Veja o exemplo na figura abaixo. A linha (a) é simples, mas a (b) não é. Da mesma forma, o polígono (c) é simples, mas (d) não é.



A terceira função de processamento geométrico verifica se um ponto está contido em um polígono, conforme abaixo:



A verificação é feita imaginando uma semi-reta em qualquer direção a partir do ponto (em geral usa-se a direção horizontal positiva, como na figura). São contadas as interseções com segmentos do contorno. Se o número de interseções for par, o ponto está fora do polígono; se for ímpar, está dentro.

No TAD você deve criar um tipo (`typedef`) para representar separadamente ponto, linha e polígono, usar a definição de ponto na definição de linha e polígono, e implementar funções que permitam realizar operações sobre os dados, de acordo com a lista abaixo:

TAD Geometria:

```
void criaPonto(ponto *p, double x, double y);

// retorna TRUE se os pontos forem identicos
int pontoCoincide(ponto P, ponto Q);

void imprimePonto(ponto P);

void criaLinha(linha *l, int numVertices, ponto *vertices);

// verifica se a linha poligonal tem interseção com o polígono
int linhaInterceptaPoligono(linha L, poligono P);

// verifica se a linha é simples (sem auto-interseções)
int linhaSimples(linha L);

void criaPoligono(polígono *p, int numVertices, ponto *vertices);

// verifica se o ponto está no interior do polígono
int pontoEmPoligono(ponto P, poligono Pol);

// verifica se o polígono é simples (sem auto-interseções)
int poligonoSimples(poligono Pol);
```

Você deverá implementar um programa principal, que receberá um arquivo de dados contendo a geometria de pontos, linhas e polígonos, e executará testes topológicos entre eles, de acordo com o especificado nas linhas restantes do arquivo.

O programa principal deverá ler, da entrada padrão, o arquivo de entrada. Quando executado na linha de comando, usar:

```
> programa < entrada.txt
```

A saída deverá ser direcionada para o dispositivo de saída padrão (`stdout`).

Exemplo de arquivo de entrada (os comentários NÃO fazem parte do arquivo de entrada real):

```
5                       // número de pontos
10 5                    // x y de cada ponto, separados por espaço; um ponto por linha
12 4
13 2
2 1
1 0
3                       // número de linhas
5                       // número de vértices da linha 1
0 0                     // x y de cada vértice da linha
1 1
3 6
7 9
14 8
13 10
3                       // número de vértices da linha 2
10 17                   // x y de cada vértice da linha 2
22 38
3 0
(...)
2                       // número de polígonos
4                       // número de vértices do polígono 1
10 10                   // x y de cada vértice do polígono 1
(...)
4                       // número de testes
1 LINSIMP 3             // Teste 1: verificar se a linha 3 é simples
2 LINPOL 3 1            // Teste 2: verificar se a linha 3 intercepta o polígono 1
3 POLSIMP 1             // Teste 3: verificar se o polígono 1 é simples
4 PTOPOL 1 1            // Teste 4: verificar se o ponto 1 está dentro do polígono 1
```

Conforme o exemplo, cada teste conterá referência uma das quatro funções: LINSIMP (linha simples), LINPOL (linha intercepta o polígono), POLSIMP (polígono simples) e PTOPOL (ponto em polígono).

Exemplo de saída (uma linha para cada teste; NÃO USAR ACENTOS; usar EXATAMENTE as frases):

```
Linha 3: simples
Linha 3: nao intercepta o poligono 1
Poligono 1: nao simples
Ponto 1: fora do poligono 1
```

Note que o programa principal não poderá acessar diretamente a estrutura interna do TAD. Se necessário, acrescente novas funções ao seu TAD detalhando-as na documentação do trabalho.

O programa criado não deve conter "menus interativos" ou "paradas para entrada de comandos" (como o `system("PAUSE")` por exemplo). Ele deve apenas ler os arquivos de entrada, processá-los e gerar os arquivos de saída.

Os TPs serão corrigidos em um ambiente Linux. Portanto, o uso de bibliotecas do Windows está PROIBIDO.

O que deve ser entregue:

- Código fonte do programa em C (todos os arquivos .c e .h), bem identado e comentado.
- Arquivo executável.
- Documentação do trabalho. Entre outras coisas, a documentação deve conter, sucintamente:

    1. Introdução: descrição sucinta do problema a ser resolvido e visão geral sobre o funcionamento do programa.
    2. Implementação: descrição sobre a implementação do programa. Deve ser detalhada a estrutura de dados utilizada (de preferência com diagramas ilustrativos), o funcionamento das principais funções e procedimentos utilizados, o formato de entrada e saída de dados, compilador utilizado, bem como decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado.
    3. Estudo de Complexidade: estudo da complexidade do tempo de execução dos procedimentos implementados e do programa como um todo (notação O), considerando conjuntos de tamanho *n*.
    4. Testes: descrição dos testes realizados e listagem da saída (não edite os resultados).
    5. Conclusão: comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.
    6. Bibliografia: bibliografia utilizada para o desenvolvimento do trabalho, incluindo sites da Internet se for o caso

    Obs: Apesar desse trabalho ser simples, a documentação pedida segue o formato da documentação que deverá ser entregue nos próximos trabalhos. Um exemplo de documentação está disponível no Moodle.

Comentários Gerais:

1 Comece a fazer este trabalho logo, enquanto o problema está fresco na memória e o prazo para terminá-lo está tão longe quanto jamais poderá estar.

2 Clareza, indentação e comentários no programa também serão avaliados.

3 O trabalho é individual.

4 A submissão será feita pelo sistema online (`http://aeds.dcc.ufmg.br`).

5 Trabalhos copiados, comprados, doados, etc. serão penalizados conforme anunciado.

6 Na prova 1, um dos exercícios será sobre a implementação do trabalho. A nota do trabalho será ponderada pela nota desse exercício.

7 Penalização por atraso: $(2^d - 1)$ pontos, onde d é o número de dias de atraso.