

O problema

A cobrança de pedágio em estradas brasileiras é um tema relevante e requer respeito à transparência, fiscalização e regulamentação. No Paraná, as tarifas que podem ser cobradas são definidas no contrato de concessão, podendo ser reajustadas anualmente. A fiscalização do serviço cabe ao DER PR - Departamento de Estradas de Rodagem do Estado do Paraná. Suponha que o seu grupo foi contratado para desenvolver um sistema automático para contagem dos veículos em uma praça de pedágio. A ideia é posicionar câmeras em pontos altos, obtendo uma visão aérea do ambiente. O sistema deve interpretar as imagens, contando e classificando os veículos. Os dados gerados pelo sistema podem então ser comparados àqueles fornecidos pela concessionária. A intenção das autoridades é usar o mesmo sistema em todas as praças de pedágio do estado.

Tarefa

Cada equipe deve criar um módulo classificador para o sistema proposto. O módulo deve ter como núcleo uma função com o seguinte protótipo:

```
int contaVeiculos (Imagem* img, Imagem* bg, int contagem [N_TIPOS_DE_VEICULOS]);
```

A função recebe como parâmetros duas imagens: uma imagem de entrada¹, que mostra a praça de pedágio em um determinado instante, e uma imagem de fundo², que mostra o mesmo local sem veículos. A função deve contar e classificar os veículos presentes na imagem de entrada, usando como base apenas os dados que podem ser obtidos das imagens, assim como conhecimento específico do domínio do problema. Para que uma solução seja possível, certas simplificações serão feitas:

- Os veículos serão classificados em 4 categorias: motocicletas, veículos médios (carros e vans), veículos longos (ônibus e caminhões menores) e veículos muito longos (caminhões maiores). Não é preciso diferenciar carros de vans, nem ônibus de caminhões, nem classificar os caminhões de acordo com o número de eixos.
- Trabalharemos com imagens sintéticas: um gerador “povoa” automaticamente a imagem com veículos. Para simular variações normalmente encontradas na prática, as imagens geradas podem conter ruídos e suavizações, mas casos como chuva, neblina, e iluminação deficiente à noite não serão cobertos.
- Os veículos presentes nas imagens terão cores fortes e aspecto pouco realista. Isso é feito porque certos tons de cinza poderiam dificultar a distinção entre os veículos e o asfalto.
- As imagens processadas mostram o mesmo ponto de vista (i.e. não é preciso “cruzar” dados coletados por várias câmeras).
- O conteúdo das imagens dadas como entrada pode ser modificado pela função (isso porque a imagem original está salva em arquivo). Como exemplo, observe a Figura 1. A Figura 1(a) mostra uma imagem de fundo, com a praça de pedágio vazia. A Figura 2(a) mostra uma imagem contendo 3 motocicletas, 10 veículos médios, 6 veículos longos, e 3 veículos muito longos. Já a Figura 3 contém apenas 1 veículo de cada tipo (os casos de teste não precisam ter todos os tipos de veículos presentes).

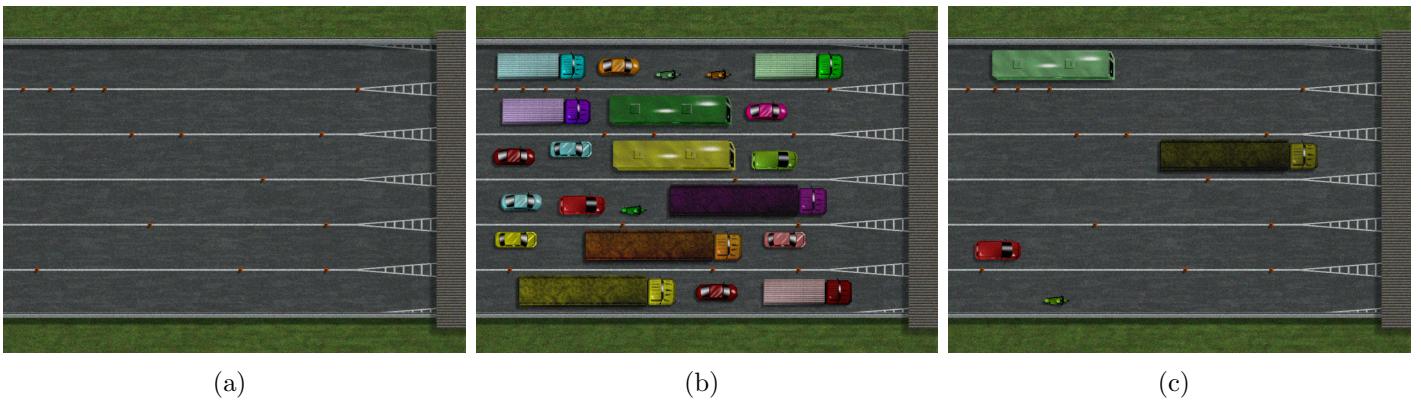


Figura 1: Exemplos. (a) praça de pedágio vazia; (b) praça de pedágio com vários veículos e (c) praça de pedágio com poucos veículos.

Para criar o trabalho, será fornecido um “pacote”³ contendo os seguintes arquivos:

1. **main.c**: Programa testador. Gera automaticamente imagens de teste e as repassa para a função central do trabalho, comparando os resultados obtidos com o que era esperado. Existem 3 constantes definidas no início do arquivo:
 - **RANDOM_SEED_OFFSET**: controla a semente do gerador de números aleatórios.
 - **N_TESTES**: número de casos de teste gerados.
 - **SALVA_INTERMEDIARIOS**: uma flag que diz se as imagens geradas devem ser salvas.
2. **imagem.c** e **imagem.h**: Funções básicas para manipulação de imagens. Inclui um tipo **Imagem**, que será usado para representar as imagens manipuladas pelo módulo classificador; e funções para criar, destruir, ler e salvar imagens. O programa testador e o módulo gerador de testes já fazem a leitura e escrita das imagens necessárias, portanto você só precisará usar as funções de manipulação de imagens se desejar criar imagens ou salvar resultados intermediários para testes/depuração.
3. **gerador_de_testes.c** e **gerador_de_testes.h**: módulo gerador de testes. é invocado pelo programa testador - você não precisa se preocupar com o que existe dentro desses arquivos.
4. **trabalho3.h**: arquivo de cabeçalho, que contém declarações para a função central do trabalho e para a constante **N_TIPOS_DE_VEICULOS**.
5. Imagens do gerador (diretório **gen**) - são as imagens usadas pelo gerador de testes. Não mexa nelas!

Para realizar o trabalho, crie um projeto, e coloque todos os arquivos de programa acima no diretório do mesmo. Mantenha o diretório **gen** como um sub-diretório do diretório do projeto. Inclua todos os arquivos de programa (.c e .h) no projeto. Após certificar-se que o programa compila e roda com o classificador “dummy” (que terá um desempenho sofrível!), o seu objetivo será criar novas versões para as funções e para o tipo definidos nos arquivos **class vei.c** e **class vei.h**. Note que o gerador de testes é usado pela função **main** fornecida, mas não precisa ser usado pelo seu classificador. As funções para manipulação de imagens só precisam ser usadas caso você deseje salvar resultados intermediários do seu classificador.

¹na verdade um ponteiro para a struct que representa a imagem de entrada

²na verdade um ponteiro para a struct que represente a imagem de fundo

³Desenvolvido pelo professor Bogdan T. Nassu

Equipes

O trabalho deve ser feito em equipes de 3 pessoas. Cada equipe deve apresentar sua própria solução para o problema. Indícios de fraude (cópia) podem levar à anulação dos trabalhos.

Entrega

O prazo de entrega é 04/07/2019, 23:59. Trabalhos entregues após esta data não serão considerados. Cada equipe deve entregar, através da página da disciplina no Moodle, um arquivo comprimido em formato .zip, contendo:

1. Um arquivo chamado `prjFinal-x-y-z.c`, onde `x`, `y` e `z` são os números de matrícula dos alunos. O arquivo deve conter a implementação da função `contaVeiculos` (com cabeçalho idêntico ao especificado), assim como de quaisquer outras funções auxiliares usadas no trabalho. Funções da biblioteca-padrão também podem ser usadas. Os autores do arquivo devem estar identificados no início, através de comentários. Separe as sub-tarefas em funções, para organizar o seu código.

IMPORTANTE: as funções pedidas não envolvem interação com usuários. Elas não devem imprimir mensagens (por exemplo, através da função `printf`), nem bloquear a execução enquanto esperam entradas (por exemplo, através da função `scanf`). O arquivo também não deve ter uma função `main`.

2. Um arquivo no formato pdf chamado `prjFinal-x-y-z.pdf`, onde `x`, `y` e `z` são os números de matrícula dos alunos. Este arquivo deve conter um relatório breve, descrevendo em detalhes a contribuição de cada membro da equipe, o funcionamento das funções entregues (explique como e por que o seu trabalho funciona), os desafios encontrados, e a forma como eles foram superados. O relatório deve ser feito em L^AT_EX, seguindo o modelo da SBC.
3. Um arquivo de texto com o nome `auto-avaliacao.txt`, o qual indica quais os requisitos que não foram cumpridos (ou que foram atendidos parcialmente). No mesmo arquivo, cada membro deve indicar quantas horas dedicou a cada etapa deste trabalho e a nota que atribuiria a si mesmo.

Avaliação

Todos os testes serão feitos usando a IDE (Integrated Development Environment) **Code::Blocks**. Certifique-se de que o seu trabalho pode ser compilado e executado a partir dela. Os trabalhos serão avaliados por meio de baterias de testes automatizados. Os testes estão implementados no arquivo `main.c` fornecido no pacote. A nota final será composta por uma nota base e descontos aplicados por conta de problemas encontrados.

Os descontos sobre a nota base serão aplicados com base nos seguintes critérios:

1. **Compilação e corretude.** O programa deve compilar para ser avaliado.
2. **Documentação.** Descreva através de comentários o funcionamento das suas funções. Não é preciso detalhar tudo linha por linha, mas forneça uma descrição geral da sua abordagem, assim como comentários sobre estratégias que não fiquem claras na leitura das linhas individuais do programa. Uma função sem comentários terá sua nota reduzida em até 25%.
3. **Indentação.** Use a indentação para tornar seu código legível. Um trabalho cujo código não esteja indentado terá sua nota reduzida em até 20%.
4. **Siga as convenções** para nomear constantes, funções e variáveis. Além disso, use nomes significativos. Um trabalho que tenha identificadores fora das convenções pode ter sua nota reduzida em até 20%.
5. **Organize seu código e use funções**, tendo em mente a modularização e a legibilidade. Um trabalho monolítico ou tendo funções muito longas/confusas terá sua nota reduzida em até 30%.
6. A não inclusão do arquivo de (auto) avaliação resulta em um desconto de 10% na nota. O mesmo vale para o relatório.