Universidade Tecnológica Federal do Paraná
Departamento Acadêmico de Informática
CSF13 – Fundamentos de Programação 1
Prof. Bogdan Tomoyuki Nassu
Profa. Leyza Baldo Dorini

Trabalho 3

I) Descrição geral

A fabricante de macarrão instantâneo Nikkin está planejando implantar em suas unidades de produção um novo sistema para controle de qualidade. Você está trabalhando na empresa que está criando este sistema, e faz parte da equipe responsável pelo módulo que analisa a produção dos temperos. Os temperos são compostos por dois ingredientes principais, e o sistema deverá monitorar a quantidade de cada ingrediente para verificar se não existem problemas na regulagem das máquinas. Um dos ingredientes tem o formato de pequenas esferas, e o outro é cortado em pequenas fatias.

A sua equipe decidiu resolver o problema capturando imagens de amostras da produção, usando uma lente com *zoom* e iluminação especial. As imagens coletadas pelo sistema são monocromáticas, e os dois ingredientes aparecem nas imagens como círculos e retângulos de tamanho variado. As Figuras 1 e 2 mostram exemplos de imagens coletadas pelo protótipo do sistema. Note que as imagens contêm ruídos, e que o formato dos ingredientes por vezes não é perfeito.

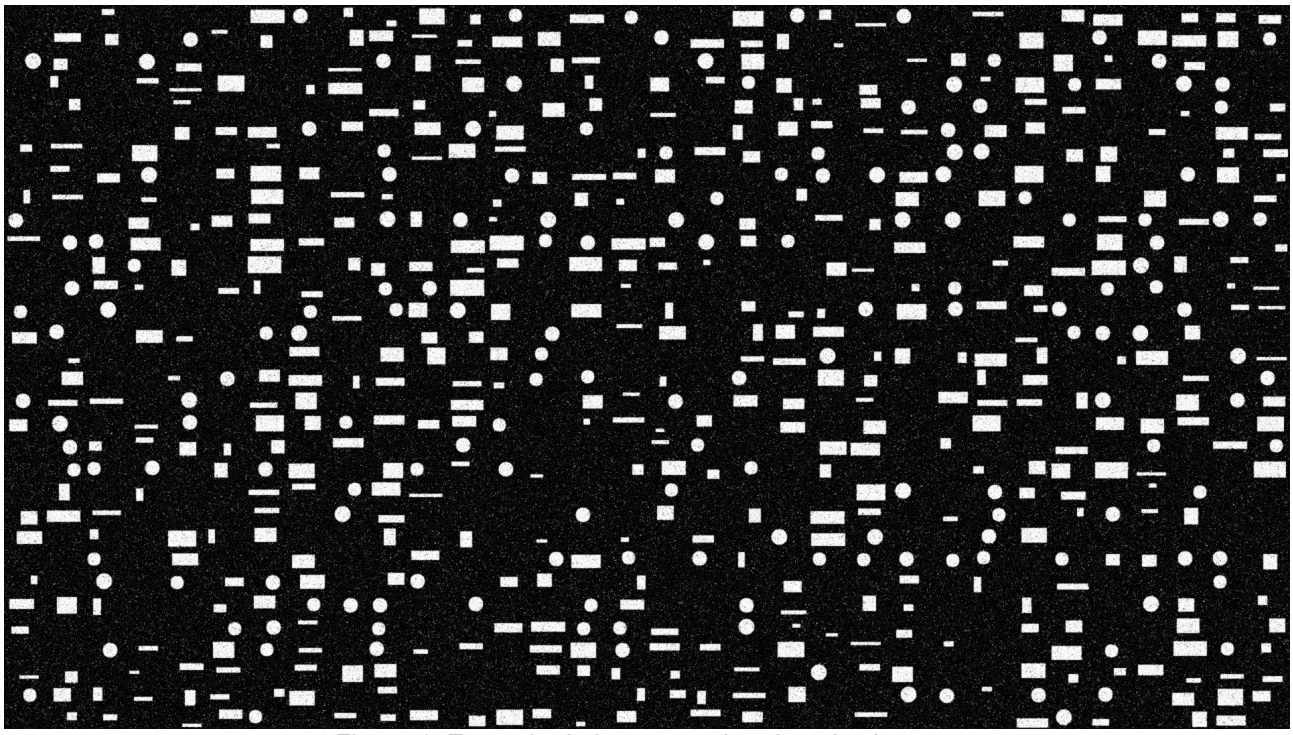

Figura 1: Exemplo de imagem coletada pelo sistema.

A sua tarefa neste projeto é desenvolver o módulo que analisa as imagens coletadas, contando os ingredientes de cada tipo observados. Mais especificamente, você deve escrever uma função com o seguinte protótipo:

```
void contaIngredientes (Imagem1C* img, int* n_circulos, int* n_retangulos);
```

A função recebe como parâmetro de entrada um ponteiro para uma imagem, e deve contar quantos ingredientes de cada tipo (“círculos” e “retângulos”) estão presentes na imagem, armazenando os resultados nos dois parâmetros de saída, `n_circulos` e `n_retangulos`, que são passados por referência. A imagem original de entrada pode ser modificada pela função (i.e. o conteúdo da imagem pode ser destruído).

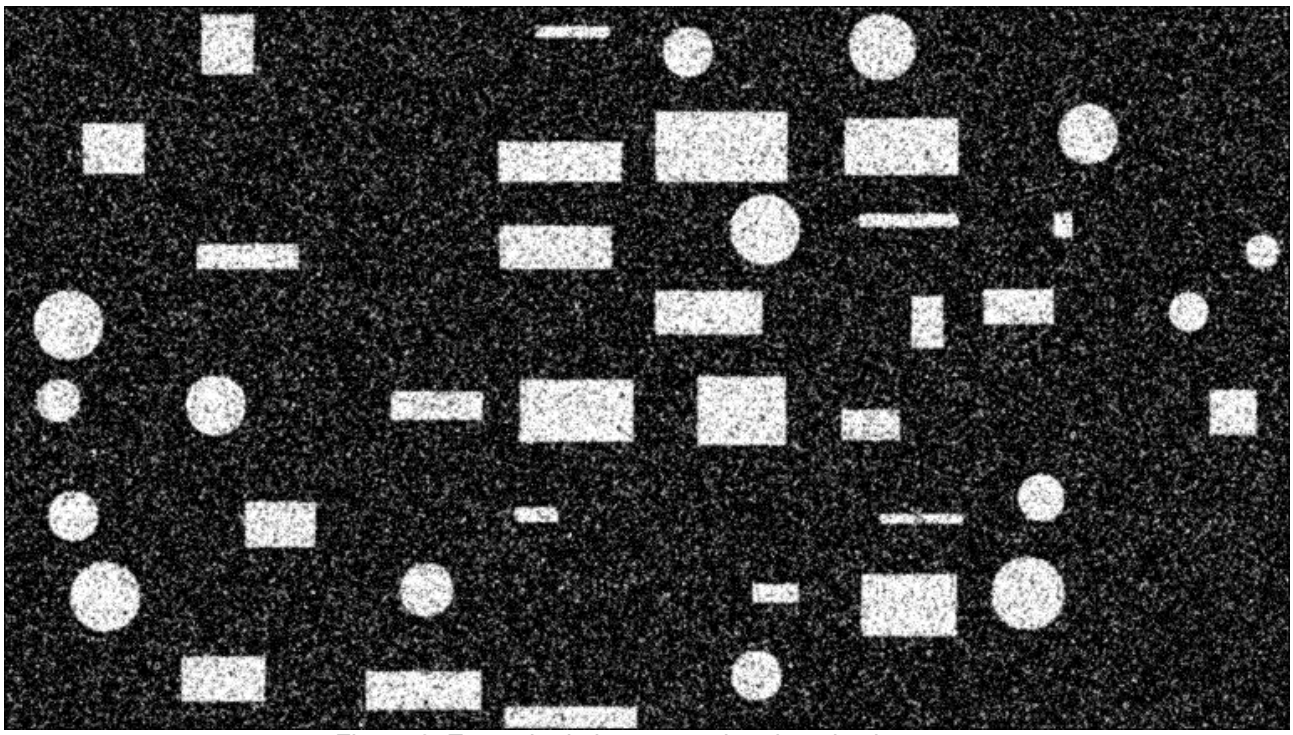

Figura 2: Exemplo de imagem coletada pelo sistema.

No momento, a sua equipe ainda não definiu qual será o sensor usado para coletar as imagens. Por isso, a sua função deve ser capaz de lidar com imagens de diferentes resoluções, e com diferentes níveis de ruído. Além disso, ainda não se sabe qual será o nível de *zoom* usado, por isso o tamanho exato dos ingredientes não é conhecido. De fato, o tamanho de cada ingrediente pode variar dentro de uma mesma imagem, devido a variações normais na produção. Considere que a equipe definiu limites mínimos para o tamanho de cada ingrediente nas imagens: os ingredientes do primeiro tipo ("círculos") não devem aparecer com raio menor que 10 pixels; enquanto os ingredientes do segundo tipo ("retângulos") devem ter altura e largura maiores ou iguais a 5 e 10 pixels, respectivamente.

Para tornar uma solução possível, faremos algumas simplificações. A primeira é a própria saída do sistema como uma simples contagem dos ingredientes – em um sistema real, seriam consideradas estatísticas baseadas em várias amostras, assim como as características dos ingredientes. Além disso, os ingredientes retangulares nunca aparecerão rotacionados, surgindo sempre alinhados aos eixos *x* e *y* da imagem. Também supomos que não ocorrem sobreposições, existindo sempre um espaço entre dois ingredientes.

Para criar o trabalho, será disponibilizado um "pacote", contendo os seguintes arquivos:

- gerador_de_testes.c, 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.

- imagem.c e imagem.h: contêm a declaração e a implementação de tipos e funções para manipulação de imagens no formato bmp.

- trabalho3.h: contém a declaração da função `contaIngredientes`. Este arquivo deve ser incluído (através da diretiva `#include`) no arquivo contendo a implementação da sua função.

- main.c: contém uma função `main` (i.e. um programa) que testa o trabalho. No começo do arquivo, existem 4 macros, que podem ser usadas para controlar os testes:

`RANDOM_SEED_OFFSET`: é a semente usada pelo gerador de números aleatórios. Modifique este valor para gerar casos de teste diferentes. Os testes finais serão realizados com um valor diferente daquele presente no arquivo original.

`N_TESTES`: número de testes a realizar.

`SALVA_INTERMEDIARIOS`: se tiver valor diferente de 0, as imagens passadas como parâmetro para a função serão salvas.

`MOSTRA_INTERMEDIARIOS`: se tiver valor diferente de 0, mostra na tela os resultados de cada teste, do contrário mostra somente os resultados finais.

A montagem do projeto, a forma de se representar e manipular imagens, e as estratégias que podem ser usadas para resolver o problema serão discutidos em sala de aula.

II) Equipes

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

III) Entrega

O prazo de entrega é 14/12/2022, 23:59. Trabalhos entregues após esta data terão sua nota final reduzida em 0,00025% para cada segundo de atraso. Cada equipe deve entregar, através da página da disciplina no Moodle, dois arquivos (separados, sem compressão):

- Um arquivo chamado *t3-x-y-z.c*, onde *x, y* e *z* são os números de matrícula dos alunos. O arquivo deve conter a implementação da função `contaIngredientes`. Separe as sub-tarefas em funções, para organizar o seu código. Qualquer função da biblioteca-padrão pode ser usada.

IMPORTANTE: a função pedida não envolve interação com usuários. Ela não deve imprimir mensagens (por exemplo, através da função `printf`), nem bloquear a execução enquanto espera entradas (por exemplo, através da função `scanf`). O arquivo também não deve ter uma função `main`.

- Um arquivo no formato PDF chamado *t3-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) a contribuição de cada membro da equipe, (b) detalhes extras sobre o funcionamento das funções entregues que não tenham ficado claros nos comentários, (c) os desafios encontrados, e (d) a forma como eles foram superados. Não é preciso seguir uma formatação específica.

Os autores dos arquivos devem estar identificados no início dos mesmos.

IV) Avaliação (normal)

Todos os testes serão feitos usando a IDE 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 pelo professor. A nota final será composta por uma nota base e descontos aplicados por conta de problemas encontrados. A nota base será decidida por uma “competição” envolvendo todos os trabalhos entregues, além de uma implementação feita pelo professor, sem otimizações ou “truques” sofisticados. As notas específicas serão atribuídas com base na colocação de cada trabalho na competição. Todos os trabalhos que obtiverem desempenho superior à implementação do professor terão a nota base superior a 100. Se vários trabalhos tiverem desempenho similar, os parâmetros do gerador de testes poderão ser modificados, de forma a produzir imagens de pior qualidade.

O desempenho será medido objetivamente, com base em estatísticas computadas a partir de certos critérios:

a) O número de erros de contagem.
b) O número de erros de classificação.
c) O tempo de execução.

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

IV.a) Compilação e corretude. Cada erro que impeça a compilação do arquivo ou que cause um erro de execução será corrigido pelo professor. Erros cuja correção seja complexa demais serão tratados como

fracassos para os casos de teste afetados. Um trabalho que exija *n* correções terá sua nota base multiplicada por 0.9^n. Certifique-se que seu código compila!!!

IV.b) Atendimento da especificação. Serão descontados até 10 pontos para cada item que não esteja de acordo com esta especificação, tais como nomes de arquivos e funções fora do padrão.

IV.c) 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%.

IV.d) Organização e legibilidade. Use a indentação para tornar seu código legível, e mantenha a estrutura do programa clara. Evite construções como *loops* infinitos terminados somente por `break`, ou *loops* `for` com varías inicializações e incrementos, mas sem corpo. Evite também replicar blocos de programa que poderiam ser melhor descritos por repetições. Um trabalho desorganizado ou cuja legibilidade esteja comprometida terá sua nota reduzida em até 30%.

IV.e) Variáveis com nomes significativos. Use nomes significativos para as variáveis – lembre-se que `n` ou `x` podem ser nomes aceitáveis para um parâmetro de entrada que é um número, mas uma variável representando uma "soma total" será muito melhor identificada como `soma_total`, `soma` ou `total`; e não como `t`, `aux2` ou `foo`. Uma função cujas variáveis internas não tenham nomes significativos terá sua nota reduzida em até 20%. Siga as convenções para nomear constantes, funções e variáveis.

IV.f) 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 25%.

V) Avaliação (especial)

Indícios de fraude ou de divisão desigual do trabalho podem levar a uma avaliação especial, com os alunos sendo convocados e questionados sobre aspectos referentes aos algoritmos usados e à implementação. Além disso, alguns alunos podem ser selecionados para a avaliação especial, mesmo sem indícios de fraude, caso exista uma grande diferença entre a nota do trabalho e das provas, ou se a explicação sobre o funcionamento de uma função no relatório for pouco clara.