Universidade Tecnológica Federal do Paraná
Departamento Acadêmico de Informática
ICSF13 – Fundamentos de Programação 1
Prof. Bogdan Tomoyuki Nassu
Profa. Leyza Baldo Dorini
Prof. Daniel Fernando Pigatto

Trabalho 1

Esta é a especificação do primeiro trabalho da disciplina ICSF13 - Fundamentos de Programação 1, profs. Bogdan T. Nassu, Leyza B. Dorini e Daniel F. Pigatto, para o período 2023/2.

I) Equipes

O trabalho é individual. A discussão entre colegas para compreender a estrutura do trabalho é recomendada e estimulada, mas cada um deve apresentar suas próprias soluções para os problemas propostos. Indícios de fraude (cópia) podem levar à avaliação especial (ver item IV). Não compartilhe códigos (fonte nem pseudocódigos)!!!

II) Entrega

O prazo de entrega é 09/10/2023. Trabalhos entregues após esta data terão sua nota final reduzida em 0,00025% para cada segundo de atraso. Cada estudante deve entregar, através da página da disciplina no Classroom, dois arquivos (separados, sem compressão):

- Um arquivo chamado *t1-x.c*, onde *x* é o seu número de matrícula. O arquivo deve conter as implementações das funções pedidas (com o cabeçalho idêntico ao especificado). Exceto afirmação em contrário, as funções pedidas não devem fazer uso de funções da biblioteca-padrão. Você também não pode usar vetores, matrizes ou outros recursos da linguagem C que não tenham sido explorados em aula antes da especificação deste trabalho. O autor do arquivo deve estar identificado no início, através de comentários.

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`.

- Um arquivo no formato PDF chamado *t1-x.pdf*, onde *x* é o seu número de matrícula. Este arquivo deve conter um relatório breve (em torno de 1 a 2 páginas), descrevendo os desafios encontrados e a forma como eles foram superados. Não é preciso seguir uma formatação específica. O autor do arquivo deve estar identificado no início.

III) 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. A referência será uma implementação criada pelos professores, sem “truques” ou otimizações sofisticadas. Os resultados dos trabalhos serão comparados àqueles obtidos pela referência. Os seguintes pontos serão avaliados:

III.a) Compilação. Cada erro que impeça a compilação do arquivo implica em uma penalidade de até 20 pontos. Certifique-se que seu código compila!!!

III.b) Corretude. Sua solução deve produzir o resultado correto para todas as entradas testadas. Uma função que produza resultados incorretos terá sua nota reduzida. Quando possível, o professor corrigirá o código até que ele produza o resultado correto. A cada erro corrigido, a nota será multiplicada por um valor: até 0.7 se o erro for facilmente detectável em testes simples, ou 0.85 do contrário.

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

III.d) Documentação. Comente a sua solução. 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%.

III.e) 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 bloco de comandos. Evite também replicar trechos de programa que poderiam ser melhor descritos por repetições. Um trabalho desorganizado ou cuja legibilidade esteja comprometida terá sua nota reduzida em até 30%.

III.f) 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%.

III.g) Desempenho. Se a estrutura lógica de uma função for pouco eficiente, por exemplo, realizando muitas operações desnecessárias ou redundantes, a sua nota pode ser reduzida em até 15%.

IV) Avaliação (especial)

Indícios de fraude podem levar a uma avaliação especial, com o aluno sendo convocado e questionado 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 a qualidade das soluções apresentadas em atividades anteriores, ou se a explicação sobre o funcionamento de uma função for pouco clara.

V) Nota

A nota final do trabalho será igual à soma das notas de cada função. A nota máxima de cada função é listada junto à sua descrição. A nota máxima para o relatório é 5 pontos.

VI) Apoio

Caso tenha dúvidas, procure a monitoria. Os professores também estarão disponíveis para tirar dúvidas a respeito do projeto, nos horários de atendimento previstos (e em casos excepcionais, fora deles). A comunicação por e-mail pode ser usada para pequenas dúvidas sobre aspectos pontuais.

Instruções para "montar" o projeto:

Será disponibilizado um "pacote" contendo os seguintes arquivos:

1) Um arquivo main.c, contendo o código-fonte de um programa para testar as funções pedidas. Você pode modificar os valores das macros `TESTA_X` no começo do arquivo, caso queira deixar de testar uma função. Note que você não precisa compreender os testes.

2) Um arquivo trabalho1.h, contendo as declarações das funções pedidas e de funções auxiliares. Este arquivo deve ser incluído no seu arquivo .c através da diretiva `#include`. As funções declaradas no arquivo se dividem em 3 grupos:

- Funções do trabalho: estas são as funções pedidas, que cada um deve implementar.

- Funções auxiliares que devem ser chamadas pelos alunos: estas são as funções mencionadas na descrição das funções pedidas. Você deve fazer uso destas funções.

- Funções auxiliares que não devem ser chamadas pelos alunos: estas funções são usadas nos testes, mas não devem ser usadas no seu trabalho.

3) Um arquivo trabalho1.c, contendo as implementações das funções auxiliares, assim como dados internos usados para os testes. Você não precisa compreender o conteúdo deste arquivo, nem mesmo das funções auxiliares usadas pelo seu trabalho. Lembre-se: é preciso abstrair!

4) Arquivos .txt contendo dados de teste. É essencial lembrar que as suas funções devem ser genéricas, e devem cobrir não apenas os casos de teste, mas qualquer outro caso que se encaixe na especificação. A correção final usará arquivos diferentes daqueles disponibilizados para testes.

Para usar os arquivos disponibilizados no Code::Blocks:

1) Crie no Code::Blocks um projeto vazio (Empty project).

2) Coloque todos os arquivos disponibilizados no mesmo diretório criado para o projeto.

3) Adicione ao projeto os arquivos main.c, trabalho1.c e trabalho1.h.

4) Crie um arquivo para as suas funções, e adicione-o ao projeto.

5) Crie corpos "vazios" para as funções pedidas. Os protótipos podem ser copiados/colados diretamente do arquivo trabalho1.h. Use valores de retorno arbitrários, por enquanto.

6) Desative todos os testes, mudando para 0 os valores de todas as macros no início do arquivo main.c.

7) Verifique se o projeto pode ser compilado e o programa executado.

8) Ative os testes um por um, para testar as funções conforme elas forem sendo implementadas.

Se você quiser utilizar outra ferramenta, pode fazê-lo, mas precisará descobrir sozinho como criar um programa a partir de múltiplos arquivos.

Função 1 (40 pontos)

```
int calculaInterseccao (int n_retangulos);
```

Muitas empresas estão atualmente buscando meios de explorar a grande quantidade de informações pessoais coletadas por sites da Web e dispositivos móveis. Esta busca vem acompanhada de muita discussão sobre o direito à privacidade e sobre os limites do que pode ou não pode ser vendido a terceiros. Até mesmo veículos destinados ao público em geral, como jornais e revistas, já publicaram matérias sobre o assunto, normalmente associado a termos como "*big data*" ou "ciência de dados".

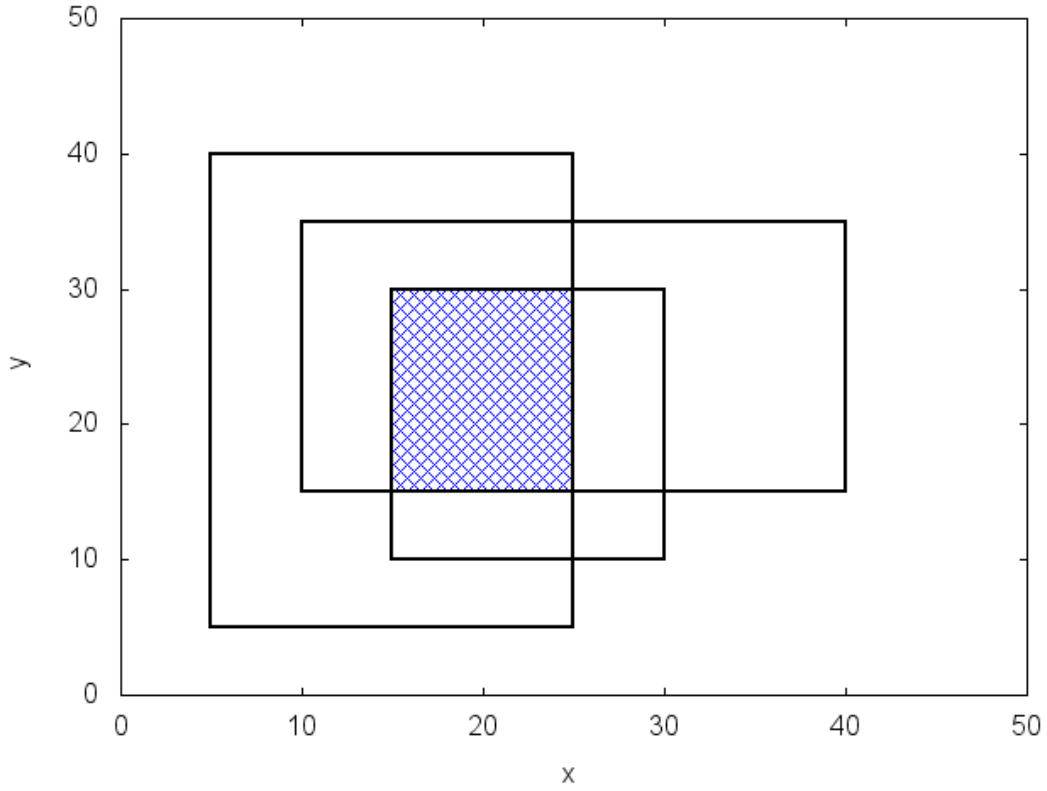
Suponha que a empresa na qual você trabalha está desenvolvendo um sistema que cruza dados de origens diversas com o propósito de localizar regiões apropriadas para a instalação de pontos de comércio e centrais de distribuição de produtos. Os dados poderiam se referir a perfis de consumidores (faixa etária, poder aquisitivo, etc.), características locais (trânsito, concorrência, etc.), ou mesmo a locais frequentados. Uma das funcionalidades do sistema é, dada uma lista de regiões que atendem a certos requisitos definidos pelo usuário, calcular e apresentar o tamanho da intersecção das regiões.

Por simplicidade, suponha que as regiões são todas retangulares. Também suponha que o sistema de coordenadas é definido sobre um plano cartesiano com origem em um ponto qualquer – em sistemas que trabalham com coordenadas geográficas, normalmente a origem é o ponto com latitude e longitude 0, mas do ponto de vista deste trabalho, isto não é importante. Cada retângulo é definido por 2 coordenadas: o ponto superior esquerdo (SE) e o ponto inferior direito (ID). Cada coordenada é definida por 2 valores inteiros: x e y. A sua tarefa é escrever uma função que coleta os dados de vários retângulos e retorna a área total do retângulo definido como a intersecção deles.

Por exemplo, suponha que o usuário definiu 3 regiões:

Região 0: SE = (5, 40), ID = (25, 5)
Região 1: SE = (10, 35), ID = (40, 15)
Região 2: SE = (15, 30), ID = (30, 10)

A figura a seguir ilustra as 3 regiões:



Neste exemplo, a intersecção é um retângulo com SE = (15, 30) e ID = (25, 15). Portanto, a área da intersecção é 150.

A função `calculaInterseccao` recebe como parâmetro apenas o número de retângulos. Para recuperar as coordenadas de cada retângulo, são fornecidas as seguintes funções auxiliares, declaradas no arquivo trabalho1.h:

```
int pegaXSE (int pos);
int pegaYSE (int pos);
int pegaXID (int pos);
int pegaYID (int pos);
```

Estas funções retornam o valor da coordenada em um dos eixos e um dos pontos. O parâmetro `pos` diz de qual retângulo a coordenada será obtida, com a contagem começando em 0. No exemplo anterior, portanto, a função `calculaInterseccao` receberia como parâmetro `n_retangulos` = 3, e os retângulos seriam numerados 0, 1 e 2. Uma chamada a `pegaXSE (0)` retornaria a posição x do ponto superior esquerdo do primeiro retângulo (5), uma chamada a `pegaYSE (1)` retornaria a posição y do ponto superior esquerdo do segundo retângulo (35), e uma chamada a `pegaXID (2)` retornaria a posição x do ponto inferior direito do terceiro retângulo (30).

Note que você não precisa compreender o funcionamento interno das funções auxiliares para usá-las. Você deve implementar apenas a função `calculaInterseccao`, que retorna a área total da intersecção. No exemplo anterior, a função retornaria o valor 150. Se não houver intersecção entre os retângulos, a função deve retornar 0.

É possível evitar a análise de todos os retângulos quando 2 retângulos já analisados não têm intersecção. O seu código deve incluir esta otimização, que deve ser explicada claramente em comentários no código.

Função 2 (55 pontos)

```
unsigned int encontraParMaisProximo (int n_retangulos);
```

Seguindo a história apresentada no problema anterior, outra funcionalidade solicitada à sua equipe é uma função que será usada para analisar o impacto da distância entre fornecedores de um mesmo produto e os lucros gerados pelos mesmos. A sua parte do trabalho envolve localizar o par de fornecedores mais próximos entre si. Como a área de influência de cada fornecedor é mais importante que a sua localização geográfica, a distância será estimada a partir da região atendida por cada fornecedor.

Novamente, por simplicidade, a região atendida por um fornecedor é representada por retângulos em um plano cartesiano, com coordenadas acessadas por meio das funções `pegaXSE`, `pegaYSE`, `pegaXID` e `pegaYID`. A posição de um fornecedor é estimada como sendo o centro do retângulo que descreve a área atendida. Desta forma, um fornecedor cujo retângulo é definido com SE = (10, 5) e ID = (21, 6) tem sua posição estimada em (15.5, 5.5). Note que, embora os retângulos tenham sido definidos como números inteiros, a posição pode ter valores não-inteiros.

A função `encontraParMaisProximo` recebe como parâmetro somente o número total de retângulos que será analisado, cada um representando um fornecedor. Ela deve localizar o par de retângulos cujos centros têm a menor distância entre si. Por exemplo, se os retângulos mais próximos entre si forem o sexto e o décimo segundo, a função deve retornar os números 5 e 11 (lembre-se que o primeiro retângulo é identificado pelo número 0).

A forma “elegante” de se retornar 2 valores envolveria uma *passagem de parâmetros por referência*. Como este assunto não foi abordado na disciplina antes da disponibilização desta especificação, usaremos um artifício pouco realista. Para retornar 2 valores, aproveitaremos os bits do valor de retorno da seguinte forma: os 16 bits mais significativos (“mais à esquerda”) recebem o número do primeiro retângulo. Os 16 bits restantes, menos significativos, recebem o número do segundo retângulo. Os bits mais significativos devem sempre identificar o retângulo com o menor identificador. Desta forma, para o exemplo com os números 11 e 5, o valor de retorno seria: 0x0005000B (= 327691 decimal, ou 00000000000001010000000000001011 binário).

Para criar a sua solução, você não pode usar vetores, matrizes ou outras estruturas de dados que não tiverem sido explicadas em aula. Se julgar necessário, você pode criar funções auxiliares. Para o cálculo das distâncias, você pode usar a função `sqrt`, declarada no arquivo math.h (consulte a documentação para saber o que ela faz e como usá-la). Outro recurso que pode ser útil é a macro `FLT_MAX`, declarada no arquivo float.h.