# Trabalho Prático 2
# Sistema de Despacho de Transporte por Aplicativo

## 1 Descrição do problema

A empresa multinacional de transporte por aplicativo *CabAI* resolveu abrir a sua filial no Brasil, denominada *CabeAí*. Uma inovação que a empresa quer ofertar no Brasil é a sugestão de corrida compartilhada, uma lógica parecida com o táxi lotação.
Você foi contratado pela *CabeAí* para implementar o sistema de despacho de corridas, incluindo a sugestão de corrida compartilhada. Na prática, esse sistema de despacho vai, diante de uma demanda (i.e., origem, destino e quando iniciar a corrida) de um cliente, não apenas selecionar o veículo a realizar o transporte, mas investigar potenciais compartilhamentos e oferecê-los em tempo real aos clientes, com o incentivo que as corridas compartilhadas tem um custo menor, por cliente, para a corrida contratada.
O grande desafio é como identificar demandas por corridas que possam ser compartilhadas. Numa primeira versão, assumimos que há apenas um passageiro por demanda por corrida individual e o compartilhamento de corrida se baseia na satisfação conjuntiva de um conjunto de critérios:

**Capacidade dos veículos:** A capacidade dos veículos, identificada por $\eta$, define o número máximo de passageiros que pode participar de uma corrida compartilhada, ou seja, define também o máximo de corridas que podem ser combinadas. Todos os veículos da frota tem a mesma capacidade.

**Velocidade dos veículos:** A velocidade dos veículos, identificada por $\gamma$, é a razão entre unidades de espaço e tempo percorridas. Ela é utilizada para calcular a duração de um trecho durante a simulação. No âmbito da simulação pretendida, todos os veículos andam na mesma velocidade.

**Intervalo entre partidas:** O intervalo entre partidas, identificado por $\delta$, identifica quais demandas por corridas podem ser combinadas com a demanda corrente. A partir da primeira corrida a ser eventualmente compartilhada, todas as corridas que ocorram nos próximos $\delta$ unidades de tempo devem ser verificadas com relação aos demais critérios.

**Distância entre origens:** O parâmetro $\alpha$ define a distância máxima entre pares de origens para que as respectivas corridas possam ser combinadas. Sem perda de generalidade, a distância entre origens é conjuntiva, ou seja, as distâncias entre as origens de todos os pares de corridas a serem compartilhadas devem ser menores que $\alpha$.

**Distância entre destinos:** O parâmetro $\beta$ define a distância máxima entre pares de destinos para que as respectivas corridas possam ser combinadas. Sem perda de generalidade, a distância entre os destinos é conjuntiva, ou seja, as distâncias entre os destinos de todos os pares de corridas a serem compartilhadas devem ser menores que $\beta$.

**Eficiência da corrida compartilhada:** O parâmetro $\lambda$ é o limiar mínimo de eficiência de uma corrida compartilhada. A eficiência da corrida deve ser maior que $\lambda$ (note que corridas individuais tem eficiência 100% por definição). A eficiência quantifica quanto da corrida compartilhada é associado à corrida em si e quanto é associado à coleta dos clientes. Por exemplo, suponhamos que a corrida seja compartilhada

por dois clientes, distantes 1 km tanto na sua origem quanto no seu destino, e a distância da corrida é 8km. Ou seja, a corrida total é 10km, sendo 8km a corrida propriamente dita, resultando numa eficiência de 80%.

Como mencionado, a entrada do sistema é uma sequência, ordenada temporalmente de solicitações de corrida. Você pode assumir, para fins de simulação que você conhece toda a demanda antecipadamente (ou seja, o arquivo de entrada é carregado antes de iniciar a simulação). A seguir descrevemos a construção das corridas a serem realizadas:
Para cada demanda de corrida $c_0$, ordenada cronologicamente, execute as seguintes operações:

1. Identifique as $n$ (onde $n \leq \eta$) corridas $c_1 \dots c_n$ que estejam dentro do intervalo de tempo $\delta$ a partir de $c_0$.
2. Seja $C = c_0$, o conjunto de corridas inicialmente unitário a partir do qual queremos gerar uma corrida combinada.
3. Seja $r = (Origem_{c_0}, Destino_{c_0})$, a corrida individual associada a $c_0$.
4. Para cada corrida $c_i$ que satisfaça o critério de intervalo de tempo menor que $\delta$ (i.e., $tempo_{c_i} - tempo_{c_0} < \delta$):
   (a) Verifique se $c_i$ satisfaz os critérios de distância entre origens ($\alpha$) e distância entre destinos ($\beta$) para todas as corridas em $C$. Se $c_i$ satisfizer os critérios, acrescente $c_i$ a $C$. Senão, interrompa a avaliação e conclua a definição de $r$.
   (b) Atualize a corrida $r$ associada ao conjunto $C$, ou seja, a sequência de pontos de parada a serem visitados pelo veículo na coleta e na entrega de passageiros. Na versão básica do simulador, as coletas e as entregas são na mesma ordem, que é também a ordem das demandas (i.e., $c_0, c_1, \dots c_k$).
   (c) Verifique se a eficiência da corrida $r$ satisfaz o critério $\lambda$. Se não satisfizer, remova $c_i$ de $C$ e de $r$, interrompa a avaliação e conclua a definição de $r$.
5. Adicione $r$ ao conjunto de corridas $R$.
6. Escalone o primeiro ponto de coleta da corrida $r$.

Uma vez que tenham sido definidas todas as corridas em $R$, assim como escalonadas a primeira coleta de cada uma no simulador de eventos de corrida, processamos as corridas como descrito a seguir.
Para cada evento $e_i$, tomado em ordem cronológica, do simulador de eventos de corrida, faça:

1. Recupere a corrida $r \in R$ associada a $e_i$.
2. Se $e_i$ for o último evento de $r$, gere as estatísticas de $r$.
3. Senão, escalone o próximo evento $e_j$ que sucede $e_i$ em $r$.

A simulação acaba quando todas as demandas definidas no arquivo de entrada tiverem sido executadas.
A seguir apresentamos informações para facilitar a implementação do simulador de eventos de corridas.

# 2 Simulação de eventos discretos

Segundo a Wikipedia [1], a simulação de eventos discretos (SED) modela a operação de um sistema como uma sequência de eventos discretos no tempo. Cada evento ocorre em um determinado instante de tempo e marca uma mudança de estado no sistema. Entre eventos consecutivos, considera-se que o sistema não sofre mudança alguma, assim, a simulação pode saltar diretamente do instante de ocorrência de um evento para o próximo.
Esta forma de execução se contrasta com a simulação contínua, na qual a simulação acompanha continuamente a dinâmica do sistema ao longo do tempo, sem saltos discretos de um evento ao outro. Assim, na simulação contínua o tempo é quebrado em pequenos intervalos e o estado do sistema é avaliado de acordo com o que ocorre dentro de cada intervalo. Em contraponto, a simulação de eventos discretos não precisa simular cada fatia de tempo e, ao saltar de um evento ao outro, ela é usualmente executada com mais rapidez que sua correspondente simulação contínua.
Uma técnica conhecida para execução de simulações de eventos discretos é o "Método das três fases". Nesta abordagem, a primeira fase sempre avança o relógio para o próximo evento a ocorrer, respeitando a ordem cronológica de eventos (chamados de eventos do tipo A). A segunda fase é a execução de todos os eventos que incondicionalmente ocorrem no instante atual (chamados de eventos do tipo B). A terceira fase é a execução de todos os eventos que condicionalmente ocorrem no tempo atual (chamados eventos do tipo C). O método das três fases é um refinamento da abordagem baseada em eventos, na qual os eventos simultâneos são ordenados de modo a tornar mais eficiente o uso dos recursos computacionais. O método das três fases é utilizado por diversos pacotes comerciais de simulação, mas do ponto de vista do usuário, tais especificidades técnicas do método de execução são geralmente ocultas.

## 2.1 Exemplo

Um exercício comum para se entender como são construídos modelos de simulação de eventos discretos é a modelagem de um sistema de fila de atendimento, tal como a fila formada por clientes que chegam em um agência bancária e esperam por atendimento no caixa. Neste caso, as entidades do sistema são os clientes que buscam atendimento e os eventos são: a chegada de um novo cliente, o início do atendimento no caixa e o fim do atendimento (equivalente à saída do cliente do sistema). Os estados do sistema, que são passíveis de alteração pelos eventos anteriores, são: o número de clientes na fila de atendimento (um número inteiro entre 0 e n) e o estado do caixa (livre ou ocupado). As variáveis aleatórias que devem ser identificadas para modelar a componente estocástica do sistema são: o tempo entre chegadas sucessivas de clientes e o tempo de serviço no caixa de atendimento.

## 2.2 Componentes

Além da lógica do que acontece quando ocorrem eventos no sistema, a simulação de eventos discretos ainda inclui os componentes descritos a seguir.

### 2.2.1 Estado

Cada estado do sistema é representado por um conjunto de variáveis que capturam as suas propriedades mais significativas. O comportamento dos estados ao longo do tempo, $S(t)$

[1] https://pt.wikipedia.org/wiki/Simulação_de_eventos_discretos

pode ser matematicamente representado por uma função degrau cujos valores mudam em resposta à execução dos eventos discretos do próprio sistema.

### 2.2.2 Relógio

A simulação deve manter controle do tempo atual de simulação, independentemente da unidade de medida de tempo utilizada pelo sistema sendo modelado. Na simulação de eventos discretos, em contraponto à denominada simulação em tempo real, o relógio "salta´´ entre os instantes de ocorrência dos eventos, ou seja, de outra forma, o relógio avança para o instante de início do próximo evento enquanto a simulação é executada.

### 2.2.3 Lista de eventos

Usualmente, a simulação mantém ao menos uma lista de eventos pendentes, que representa os eventos que ainda devem ser executados. Um evento é descrito pelo seu instante de ocorrência no tempo e um tipo, indicando o código que será usado para simular o evento. É comum que o código do evento seja parametrizado e, neste caso, a descrição do evento também conterá parâmetros para o seu código de execução.
Quando os eventos são instantâneos, as atividades que se estendem ao longo do tempo são modeladas como sequências de eventos. Alguns ambientes de simulação permitem que o instante de execução de um evento possa ser especificado por um intervalo identificado pelo instante de início e fim do evento.
O conjunto de eventos aguardando por execução são usualmente organizados como uma fila com prioridade, ordenada cronologicamente. Isto é, independentemente da ordem com que os eventos são adicionados ao conjunto de eventos, eles são removidos em ordem estritamente cronológica. Diversos algoritmos genéricos de busca e ordenação de filas com prioridade se provaram eficazes para a simulação de eventos discretos, tais como o *splay tree*, *skip lists*, *calendar queues*, e *ladder queues*.
Tipicamente, os eventos são agendados para execução de modo dinâmico, ao longo da própria da simulação. Por exemplo, no exemplo do banco proposto anteriormente, o evento "chegada do cliente´´ no instante $t$ poderia, caso a fila de clientes esteja vazia e o caixa livre neste instante, incluir a criação dos eventos subsequentes "saída do cliente´´ no instante $t + s$, onde $s$ é um tempo gerado a partir da distribuição do tempo de serviço no caixa.

### 2.2.4 Estatísticas

Uma simulação normalmente estima as principais estatísticas de interesse do comportamento do sistema. No caso do banco, por exemplo, uma estatística de interesse seria o tempo médio de espera por atendimento dos clientes. Em um modelo de simulação, as métricas (como a do tempo médio de espera por atendimento) são geralmente obtidas por meio de médias de replicações do modelo. Cada replicação representa uma diferente execução completa do modelo. Para se avaliar a qualidade do valor obtido, são construídos Intervalos de confiança para as estatísticas estimadas.

### 2.2.5 Condições de finalização de execução

Teoricamente, uma simulação de eventos discretos poderia ser executada para sempre. Assim, o projetista do modelo de simulação deve decidir quando a simulação deve terminar. Tipicamente, as condições de finalização são "no instante $t$´´ ou "após o processamento de

```
Inicializar Condição de Término para FALSO
Inicializar as variáveis de estado do sistema
Inicializar o Relógio (usualmente zero)
Agendar um evento inicial
Enquanto (a condição de término é FALSA)
    Definir o relógio para o instante do próximo evento
    Executar o próximo evento
    Se for o caso, agendar novo(s) evento(s)
    Remover o evento executado da lista de eventos
    Atualizar as estatísticas
Fim
Gerar relatórios de estatísticas
```

Figura 1: Pseudo-código de simulador discreto de eventos típico

```
<eta> // capacidade do veículo
<gama> // velocidade do veículo
<delta> // intervalo temporal máximo entre corridas combinadas
<alfa> // distância máxima entre origens de corridas combinadas
<beta> // distância máxima entre destinos de corridas combinadas
<lambda> // eficiência mínima da corrida combinada
<numdemandas> // número de demandas a serem simuladas
<iddemanda1> <tempo1> <origem1> <destino1>
...
<iddemandan> <tempon> <origemn> <destinon>
```

Figura 2: Especificação do arquivo de entrada

um número $n$ de eventos´´ ou, mais geralmente, "quando a medida estatística $X$ atingir o valor $x$´´.

#### 2.2.6 Execução

Um simulador de eventos discretos possui a estrutura típica ilustrada na Figura 1.

## 3 Sistema de Despacho de Transporte por Aplicativo CabeAí

Nesta seção vamos descrever o Sistema de Despacho a ser implementado no trabalho prático.

### 3.1 Formato do arquivo de entrada

O arquivo de entrada para a sua simulação possui dois tipos de informação: (1) parâmetros da simulação; e (ii) lista de demandas e suas características. A Figura 2 apresenta a especificação do arquivo de entrada.
Já a saída é uma linha por corrida concluída, seja ela combinada ou não, contendo as seguintes informações:

1. Tempo de conclusão da corrida

2. Distância total percorrida
3. Número de paradas
4. Sequência de coordenadas associadas às paradas.

## 3.2 Tipos abstratos de dados

Para a simulação proposta, você precisa projetar e implementar 4 tipos abstratos de dados, descritos a seguir.

### 3.2.1 Demanda

A demanda por uma corrida, cujas informações são lidas do arquivo de entrada, representa a origem e destino de uma corrida, assim como um identificador e quando ela foi solicitada. Para fins da simulação, a demanda por uma corrida pode estar em 4 estados:

1. Corrida demandada
2. Corrida individual
3. Corrida combinada
4. Corrida concluída

O TAD demanda, além de manter os dados da demanda e a corrida associada, deve armazenar o estado atual, e as estatísticas de execução da corrida. Essas informações serão fundamentais para o cálculo das estatísticas gerais do sistema.

## 3.3 Parada

O TAD parada mantém as informações sobre cada parada feita por um veículo durante uma corrida. Isso inclui, além das coordenadas da parada, se ela foi de embarque ou desembarque, e qual o passageiro (de qual demanda) que embarcou ou desembarcou, respectivamente.

## 3.4 Trecho

O TAD trecho registra as informações de deslocamento de um veículo entre duas paradas. Isso inclui a referência para as duas paradas que o delimitam, o tempo, a distância percorrida e a natureza do trecho: coleta (caracterizado por duas paradas de embarque), entrega (caracterizado por duas paradas de desembarque), ou deslocamento (caracterizado por um embarque e um desembarque)

## 3.5 Corrida

O TAD corrida registra as informações da satisfação de uma ou mais demandas por corrida. Além dos identificadores das demandas satisfeitas pela corrida, inclui uma lista de trechos e outras informações como a sua duração, distância e eficiência. Importante notar que a corrida tem um e somente um trecho de deslocamento, e o mesmo número de trechos de coleta e entrega, que é igual ao número de passageiros total menos um.

#### 3.5.1 Escalonador

O escalonador é um elemento central da simulação de eventos discretos. Ele é implementado como uma fila de prioridade que recupera o próximo evento (ou seja o evento de menor data-hora que está na fila). Sugere-se implementar a fila de prioridade utilizando um *minheap*.
As operações a serem implementadas incluem:

1. Inicializa
2. InsereEvento
3. RetiraProximoEvento
4. Finaliza

Para fins de escalonamento, você pode utilizar números inteiros ou reais, a partir de uma data de referência. As estatísticas de escalonamento devem ser geradas quando finalizar.

### 3.6 Análise Experimental

A análise experimental deve exercitar dimensões que efetivamente impactam o custo computacional da simulação. A princípio distinguimos duas dimensões que podem ser avaliadas:

**Candidatos a combinação:** O número de candidatos a combinação é função de $\delta$, $\alpha$, $\beta$ e $\eta$. Note que relaxar os três primeiros parâmetros enquanto último parâmetro permanece constante pode não afetar o número de passageiros em corridas compartilhadas, e vice-versa.

**Custo de coleta e de entrega:** O custo de coleta e de entrega é função dos parâmetros $\alpha$ e $\beta$, sendo limitado pelo parâmetro $\lambda$. A redução dos três parâmetros certamente diminui a probabilidade de corridas compartilhadas, enquanto o seu aumento surte o efeito contrário. É ainda possível variá-los de forma inversamente proporcional, o que pode nos permitir, por exemplo, manter a eficiência à medida que aumenta o número de passageiros compartilhando a corrida.

Sugerimos que avalie-se cada uma das dimensões por vez. Por exemplo, um resultado da avaliação pode ser um gráfico que mostre as características e a eficiência das corridas à medida que algum dos parâmetros varia. Para cada caso, considere uma configuração base de todos os parâmetros e varie os valores da dimensão sendo avaliada. Outras dimensões podem ser consideradas na avaliação experimental.

## 4 Pontos extras

Serão distribuídos pontos extras em duas situações:

**Qualidade da solução:** Mesmo na sua versão básica, a eficiência das corridas compartilhadas pode ser melhorada, por exemplo, minimizando os custos das rotas de coleta e de entrega, assim como reduzindo o cumprimento do trecho de deslocamento. Isso pode ser feito explorando outras rotas de coleta e entrega, mudanças que também alteram o trecho de deslocamento. Uma outra limitação da geração da rota atual é

que, quando uma demanda por corrida não satisfaz algum critério, a montagem da corrida compartilhada é interrompida. Uma solução poderia ser verificar todas as demandas candidatas a serem combinadas, excluindo apenas aquelas demandas que não satisfazem os critérios. Importante fazer a análise experimental das melhorias e verificar o seu impacto em termos de melhoria dos serviços prestados. Finalmente, podemos também avaliar a utilização de ambas as estratégias e outras que sejam possíveis.

**Funcionalidades Adicionais:** Podemos imaginar outras funcionalidades adicionais que poderiam ser incluídas no portfolio de serviços da *CabeAí*:

**Corrida dinâmica:** A corrida dinâmica permite a adição de passageiros após o início da corrida, ou seja, há uma demanda, que não necessariamente satisfaz o requisito do parâmetro $\delta$, mas que demanda apenas um pequeno desvio no trecho de deslocamento da corrida combinada, seja para coletar, seja para entregar, ou ambos. Para isso seria necessário modelar por onde a rota da corrida passa e avaliar quais as corridas não combinadas que poderiam ser agregadas dinamicamente à rota em questão.

**Quase um busão:** A funcionalidade *quase um busão* é uma generalização da corrida dinâmica, pois vai permitir que vários veículos sejam usados para atender uma corrida. Por exemplo, considerando uma corrida individual, o veículo A levaria o passageiro até um ponto intermediário da corrida, a partir de onde um veículo B levaria até o destino. A solução pode ser generalizada tanto em termos do número de particionamento de uma dada corrida, quanto em termos de estratégias de compartilhamento de corridas usando múltiplos veículos, à semelhança do problema de fluxo máximo em redes.

Outras propostas de otimização são possíveis e serão valoradas. É necessário justificar, quantificar e entender os eventuais ganhos de uma proposta de otimização para receber os pontos extras.

# 5 Como será feita a entrega

## 5.1 Submissão

A entrega do TP2 compreende duas submissões:

**VPL TP2:** Submissão do código a ser submetido até **17/11, 23:59** (o sistema vai ficar aberto madrugada adentro, mais para evitar problemas transientes de infraestrutura). **Submissões em atraso terão desconto.** Detalhes sobre a submissão do código são apresentados na Seção 5.3.

**Relatório TP2:** Arquivo PDF contendo a documentação do TP, assim como a avaliação experimental, conforme instruções, a ser submetido até **17/11, 23:59** (o sistema vai ficar aberto madrugada adentro, mais para evitar problemas transientes de infraestrutura). **Submissões em atraso terão desconto.** Detalhes sobre a submissão de relatório são apresentados na Seção 5.2.

### 5.2 Documentação

A documentação do trabalho deve ser entregue em formato **PDF** e também **DEVE** seguir o modelo de relatório que será postado no `minha.ufmg`. Além disso, a documentação deve conter **TODOS** os itens descritos a seguir **NA ORDEM** em que são apresentados:

1. **Capa:** Título, nome, e matrícula.
2. **Introdução:** Contém a apresentação do contexto, problema, e qual solução será empregada.
3. **Método:** Descrição da implementação, detalhando as estruturas de dados, tipos abstratos de dados (ou classes) e funções (ou métodos) implementados.
4. **Análise de Complexidade:** Contém a análise da complexidade de tempo e espaço dos procedimentos implementados, formalizada pela notação assintótica.
5. **Estratégias de Robustez:** Contém a descrição, justificativa e implementação dos mecanismos de programação defensiva e tolerância a falhas implementados.
6. **Análise Experimental:** Apresenta os experimentos realizados em termos de desempenho computacional[2], assim como as análises dos resultados.
7. **Conclusões:** A Conclusão deve conter uma frase inicial sobre o que foi feito no trabalho. Posteriormente deve-se sumarizar o que foi aprendido.
8. **Bibliografia:** Contém fontes utilizadas para realização do trabalho. A citação deve estar em formato científico apropriado que deve ser escolhido por você.

Número máximo de páginas incluindo a capa: 10
A documentação deve conter a descrição do seu trabalho em termos funcionais, dando foco nos algoritmos, estruturas de dados e decisões de implementação importantes durante o desenvolvimento.

Evite a descrição literal do código-fonte na documentação do trabalho.

**Dica:** Sua documentação deve ser clara o suficiente para que uma pessoa (da área de Computação ou não) consiga ler, entender o problema tratado e como foi feita a solução.

A documentação deverá ser entregue como uma atividade separada designada para tal no minha.ufmg. A entrega deve ser um arquivo `.pdf`, nomeado `nome_sobrenome_matricula.pdf`, onde nome, sobrenome e matrícula devem ser substituídos por suas informações pessoais.

### 5.3 Código

Você deve utilizar a linguagem C ou C++ para o desenvolvimento do seu sistema. O uso de estruturas pré-implementadas pelas bibliotecas-padrão da linguagem ou terceiros é terminantemente vetado. Você DEVE utilizar a estrutura de projeto abaixo junto ao Makefile:

[2]Para este trabalho não é necessário analisar a localidade de referência.

```
– TP
    |– src
    |– bin
    |– obj
    |– include
    Makefile
```

A pasta **TP** é a raiz do projeto; **src** deve armazenar arquivos de código (*.c, *.cpp, ou *.cc); a pasta **include**, os cabeçalhos (headers) do projeto, com extensão *.h, por fim as pastas **bin** e **obj** devem estar vazias. O Makefile deve estar na raiz do projeto. A execução do Makefile deve gerar os códigos objeto *.o no diretório **obj** e o executável do TP no diretório **bin**. O arquivo executável **DEVE** se chamar **tp3.out** e deve estar localizado na pasta **bin**. O código será compilado com o comando:

```
make all
```

O seu código será avaliado através de uma **VPL** que será disponibilizada no moodle. Você também terá à disposição uma VPL de testes para verificar se a formatação da sua saída está de acordo com a requisitada. A VPL de testes não vale pontos e não conta como trabalho entregue. Um pdf com instruções de como enviar seu trabalho para que ele seja compilado corretamente estará disponível no Moodle.

# 6 Avaliação

- Corretude na execução dos casos de teste - (20% da nota total)
- Indentação, comentários do código fonte e uso de boas práticas - (10% da nota total)
- Conteúdo segundo modelo proposto na seção **Documentação**, com as seções detalhadas corretamente - (20% da nota total)
- Definição e implementação das estruturas de dados e funções - (10% da nota total)
- Apresentação da análise de complexidade das implementações - (10% da nota total)
- Análise experimental - (25% da nota total)
- Aderência completa às instruções de entrega - (5% da nota total)

Se o programa submetido **não compilar**[3] ou se compilar mas não passar em **pelo menos um caso de teste**, seu trabalho não será avaliado e sua nota será **0**. Trabalhos entregues com atraso sofrerão **penalização de** $2^{d-1}$ pontos, com d = dias úteis de atraso.

# 7 Considerações finais

1. Comece a fazer esse trabalho prático o quanto antes, enquanto o prazo de entrega está tão distante quanto jamais estará.
2. Leia atentamente o documento de especificação, pois o descumprimento de quaisquer requisitos obrigatórios aqui descritos causará penalizações na nota final.

[3]Entende-se por compilar aquele programa que, independente de erros no Makefile ou relacionados a problemas na configuração do ambiente, funcione e atenda aos requisitos especificados neste documento em um ambiente Linux.

3. Certifique-se de garantir que seu arquivo foi submetido corretamente no sistema.

4. Plágio é crime. Trabalhos onde o plágio for identificado serão **automaticamente anulados** e as medidas administrativas cabíveis serão tomadas (em relação a todos os envolvidos). Discussões a respeito do trabalho entre colegas são permitidas. É permitido consultar fontes externas, desde que exclusivamente para fins didáticos e devidamente registradas na seção de bibliografia da documentação. **Cópia e compartilhamento de código não são permitidos.**

# 8 FAQ (*Frequently asked Questions*)

1. Posso utilizar qualquer versão do C++? NÃO, o corretor da VPL utiliza C++11.

2. Posso fazer o trabalho no Windows, Linux, ou MacOS? SIM, porém lembre-se que a correção é feita sob o sistema Linux, então certifique-se que seu trabalho está funcional em Linux.

3. Posso utilizar alguma estrutura de dados do C++ do tipo Queue, Stack, Vector, List, etc? NÃO.

4. Posso utilizar smart pointers? NÃO.

5. Posso utilizar o tipo String? SIM.

6. Posso utilizar o tipo String para simular minhas estruturas de dados? NÃO.

7. Posso utilizar alguma biblioteca para tratar exceções? SIM.

8. Posso utilizar alguma biblioteca para gerenciar memória? SIM.

9. As análises e apresentação dos resultados são importantes na documentação? SIM.

10. Os meus princípios de programação ligados a C++ e relacionados a engenharia de software serão avaliados? NÃO.

11. Posso fazer o trabalho em dupla ou em grupo? NÃO.

12. Posso trocar informações com os colegas sobre os fundamentos teóricos do trabalho? SIM.

13. Posso utilizar IDEs, Visual Studio, Code Blocks, Visual Code, Eclipse? SIM.