Trabalho Prático 1 - Ordenador Universal

1 Descrição do problema

A empresa Zambs deseja lançar um produto revolucionário no mercado, o Tipo de Abstrato de Dados Ordenador Universal, que é capaz de escolher o(s) melhor(es) algoritmo(s) de ordenação e respectivos parâmetros entre os vários existentes, considerando as características do vetor de dados a ser ordenado. Essa escolha de algoritmos e seus parâmetros deve ser "aprendida"pelo algoritmo a partir da sua utilização em diversos cenários.
Antes de investir em uma implementação mais complexa, a empresa solicitou que você implemente uma prova de valor do TAD explorando dois critérios para escolha entre dois algoritmos, nominalmente inserção e quicksort:

1. Para vetores "quase"ordenados, o algoritmo inserção é mais rápido do que o algoritmo quicksort. Nesse caso, você pode considerar como medida de ordenação de um vetor o número de quebras, ou seja, o número de vezes que um elemento é menor que o seu antecessor imediato. Note que se um vetor de tamanho n estiver inversamente ordenado, o número de quebras é $n - 1$.

2. Para vetores cujo tamanho seja menor que um limiar, o algoritmo inserção deve ser mais eficiente que o quickSort recursivo. Utilize mediana de 3 para evitar casos extremos de ineficiência do algoritmo.

Para cada um dos casos há um limiar a ser determinado que depende das características do vetor a ser ordenado. No caso da inserção, há um número de quebras a partir da qual devemos passar a utilizar o quicksort. No caso do quicksort com inserção, há um tamanho mínimo de partição que é razoável utilizar o quicksort.
Nessa versão inicial, a lógica do seu TAD para a escolha de algoritmo é apresentada na Figura 1.
Como mencionado, um dos maiores desafios da sua implementação é determinar os valores de `limiarQuebras` e `minTamParticao`. Essa determinação deve ser empírica, considerando o número de comparações, movimentações e chamadas de função realizadas pelo algoritmo. Sem perda de generalidade, vamos assumir que o custo de ordenação do TAD proposto possa ser expresso pela função:

$$f(cmp, move, calls) = a * cmp + b * move + c * calls;$$

onde a, b e c são coeficientes obtidos empiricamente na plataforma alvo.
Por exemplo, para determinar o limiar de tamanho de partição para invocar o algoritmo de inserção, podemos utilizar a estratégia de varredura apresentada na Figura 2.
A estratégia para determinação de limiar de partição é iterativa e considera, na primeira iteração, limiares na faixa entre 2 e o tamanho do vetor sendo ordenado. A faixa de valores é refinada sucessivamente até que uma de duas condições seja atendida. A primeira é que a diferença de custo entre os extremos da faixa `diffCusto` seja menor que o limiar definido no arquivo de entrada (`limiarCusto`). A segunda é que a faixa tenha pelo menos 5 tamanhos de partição distintos.
Cada faixa `minMPS ... maxMPS` é dividida em 5 intervalos equidistantes, os quais estão associados a 6 tamanhos de partição. Para cada tamanho de partição, executamos o `OrdenadorUniversal`, seguido do registro (`registraEstatisticas`) e impressão das estatísticas (`imprimeEstatisticas`) da sua execução.

```
  void ordenadorUniversal(vetor V, int tam,
       int minTamParticao, int limiarQuebras){
  if (numeroQuebras < limiarQuebras){
      insercao(V,tam);
  } else {
    if (tam > minTamParticao){
      quickSort(V,tam);
    } else {
      insercao(V,tam);
    }
  }
}
```

Figura 1: Estratégia para escolha de algoritmo

```
int determinaLimiarParticao(V,tam,limiarCusto){
  minMPS = 2;
  maxMPS = tam;
  passoMPS = (maxMPS–MinMPS)/5;
  while ((diffCusto > limiarCusto) && (numMPS >= 5)){
    numMPS=0;
    for (t=minMPS; t<=maxMPS; t+=passoMPS){
      OrdenadorUniversal(V, tam, t, tam);
      registraEstatisticas(custo[numMPS]);
      imprimeEstatisticas(custo[numMPS]);
      numMPS++;
    }
    limParticao = menorCusto();
    calculaNovaFaixa(limParticao,minMPS,maxMPS,passoMPS);
    diffCusto = fabs(custo[minMPS]-custo[maxMPS]);
  }
  return limParticao;
}
```

Figura 2: Estratégia para determinação do limiar de partição

```
void calculaNovaFaixa(limParticao, minMPS, maxMPS, passoMPS){
  if (limParticao==0){
    newMin = 0;
    newMax = 2;
  } else if (limParticao==numMPS-1){
    newMin = numMPS-3;
    newMax= numMPS-1;
  } else {
    newMin = limParticao-1;
    newMax= limParticao+1;
  }
  minMPS = getMPS(newMin);
  maxMPS = getMPS(newMax);
  passoMPS = (int)(maxMPS-minMPS)/5;
  if (passoMPS == 0) passoMPS++;
}
```

Figura 3: Estratégia para calcular os valores da nova faixa

A partir dos valores das 6 execuções, identificamos a partição de menor custo (`menorCusto`), calculamos nova faixa (`calculaNovaFaixa`) e a diferença de custo associada à nova faixa. A estratégia continua até que o processo convirja.
Uma parte fundamental da estratégia é a função `calculaNovaFaixa` que recebe como parâmetro de entrada o limiar que apresentou menor custo (`limParticao`) e retorna os novos valores para a faixa.
Uma estratégia parecida pode ser utilizada para determinar o número máximo de quebras que seja razoável utilizar o algoritmo inserção. A elaboração dessa estratégia é parte do trabalho prático.

2 Entrada e Saída

O sistema a ser implementado recebe um arquivo de entrada, pela entrada padrão (`stdin`), no formato apresentado na Figura 4.
As 5 primeiras linhas da entrada definem alguns parâmetros do sistema:

1. **Limiar de Convergência:** `limiarCusto` = 10
2. **Coeficiente das Comparações:** a = 0.012156
3. **Coeficiente das Movimentações:** b = -0.00637855
4. **Coeficieinte das Chamadas:** c = 0.0172897
5. **Número de Chaves:** `tam` = 1000

A partir dessa entrada, que deve ser lida uma única vez, o sistema a ser implementado deve se calibrar (isto é, determinar os limiares de partição e de quebras) para as características da plataforma e do arquivo de entrada.
Um exemplo de saída para a determinação do limiar de partição é apresentada na Figura 4.
A primeira linha apenas informa o número de registros no arquivo de entrada. Temos então

um bloco de linhas para cada iteração. A primeira linha do bloco identifica a iteração. As próximas seis linhas do bloco apresentam as estatísticas para cada valor limiar de particao (identificado por `mps`) testado. `cost` identifica o valor estimado da ordenação, em função do número de comparações (`cmp`), número de movimentações (`move`) e número de chamadas de função (`calls`). A última linha do bloco informa quantos experimentos foram realizados (`numcost`), o limiar de partição que apresentou o melhor custo (`minmps`) e a diferença entre os novos extremos da faixa de valores a ser testada.
Um exemplo de saída para a determinação do limiar de quebras é apresentado nas Figuras 5 e 6. Novamente, a saída é dividida em blocos e cada bloco se refere a uma iteração do algoritmo (`iter`). O restante do bloco compreende 13 linhas, sendo as 12 primeiras os resultados para os diferentes limiares e a última linha apresenta algumas estatísticas sobre a iteração.
Importante ressaltar que essas instruções se aplicam à submissão VPL do TP1. Para o restante do trabalho, há toda uma análise experimental a ser realizada, descrita na Seção 3.

3 Análise Experimental

A análise experimental tem por objetivo avaliar o comportamento do TAD em relação à plataforma de execução (p.ex., seu computador pessoal) e também calibrar os vários parâmetros.
Em termos do vetor de entrada a ser ordenado temos 3 dimensões a serem avaliadas:

Configuração do vetor: Em termos de configuração os vetores a serem ordenados podem estar ordenados, inversamente ordenados, e desordenados em diversos níveis (medido pelo número de quebras). Avalie o seu TAD para várias configurações do vetor.

Tamanho da chave: O tamanho da chave é determinante para o custo de comparação. Varie o tamanho da chave para avaliar como os limiares do TAD mudam.

Tamanho do registro: O tamanho do registro é determinante para o custo de movimentação. Varie o tamanho do registro para avaliar como os limiares do TAD mudam.

Tamanho do vetor: O tamanho do vetor a ser ordenado é determinante para o custo do algoritmo como um todo.

A análise experimental deve compreender os seguintes passos:

1. Elabore o seu plano experimental, identificando os valores a serem verificados em cada dimensão.
2. Para cada configuração experimental, faça:
   (a) Calibre os custos de comparação, movimentação e chamadas de função. Isso é feito executando o TAD para diferentes tamanhos do vetor, medindo o tempo de execução e realizando a regressão para calcular os coeficientes.
   (b) Utilizando os custos calculados no item anterior, calibre o TAD em termos de limiar de partição e limiar de quebras, à semelhança do que foi feito na parte VPL do TP.
   (c) Avalie o desempenho do TAD para vários tamanhos de vetores, comparando com as versões não otimizadas dos algoritmos.

```
10                  size 1000 seed 1 breaks 502
0.012156
-0.00637855         iter 0
0.0172897           mps 2 cost 118.428736100 cmp 11772 move 7904 calls 1489
1000                mps 201 cost 214.439538500 cmp 36789 move 36560 calls 25
124                 mps 400 cost 426.090159650 cmp 74149 move 74537 calls 10
1363                mps 599 cost 724.355621100 cmp 126188 move 126934 calls 4
2504                mps 798 cost 724.355621100 cmp 126188 move 126934 calls 4
1007                mps 997 cost 724.355621100 cmp 126188 move 126934 calls 4
1696                nummps 6 limParticao 2 mpsdiff 307.661438
5
562                 iter 1
2971                mps 2 cost 118.428736100 cmp 11772 move 7904 calls 1489
2251                mps 81 cost 114.652392050 cmp 18275 move 17051 calls 73
1098                mps 160 cost 200.274470000 cmp 34209 move 33872 calls 28
1053                mps 239 cost 219.854944200 cmp 37923 move 37864 calls 22
1720                mps 318 cost 365.516470150 cmp 63459 move 63669 calls 13
397                 mps 397 cost 426.090159650 cmp 74149 move 74537 calls 10
192                 nummps 6 limParticao 81 mpsdiff 81.845734
2852
460                 iter 2
1753                mps 2 cost 118.428736100 cmp 11772 move 7904 calls 1489
649                 mps 33 cost 87.715225900 cmp 12691 move 10860 calls 157
2419                mps 64 cost 102.385287950 cmp 15960 move 14603 calls 88
421                 mps 95 cost 126.496383750 cmp 20469 move 19351 calls 64
1866                mps 126 cost 160.924912450 cmp 26937 move 26223 calls 43
632                 mps 157 cost 198.747923700 cmp 33800 move 33340 calls 31
19                  nummps 6 limParticao 33 mpsdiff 16.043447
1719
2797                iter 3
1020                mps 2 cost 118.428736100 cmp 11772 move 7904 calls 1489
2673                mps 14 cost 86.552472050 cmp 11236 move 8779 calls 345
1781                mps 26 cost 85.810712450 cmp 12027 move 10007 calls 199
1178                mps 38 cost 90.958280150 cmp 13389 move 11633 calls 139
2697                mps 50 cost 97.803628100 cmp 14962 move 13460 calls 103
2084                mps 62 cost 102.385287950 cmp 15960 move 14603 calls 88
....                nummps 6 limParticao 26 mpsdiff 4.405808
```

Figura 4: Exemplos de entrada e de saída

3. Avalie os compromissos entre as várias configurações experimentais, em particular comparando os limiares de partição e quebra obtidos.

4 Pontos Extra

Os pontos extra tem por objetivo aumentar a flexibilidade e eficência do TAD, como por exemplo:

- Adicionar novos algoritmos e estender a estratégia apresentada na Figura 1.
- Tornar o TAD realmente adaptativo, no sentido de incorporar todo o mecanismo de decisão ao TAD (aquele que foi realizado na análise experimental), ou seja, ao receber um vetor, ele verifica se há alguma "memória"em termos de uma configuração

```
iter 0
qs lq 1 cost 74.492674050 cmp 6891 move 1969 calls 190
in lq 1 cost 4.183187050 cmp 1824 move 2823 calls 1
qs lq 100 cost 87.210576300 cmp 8785 move 3544 calls 175
in lq 100 cost 350.310216550 cmp 61734 move 62733 calls 1
qs lq 199 cost 85.269510200 cmp 9265 move 4820 calls 196
in lq 199 cost 612.098030950 cmp 107046 move 108045 calls 1
qs lq 298 cost 94.594268350 cmp 10604 move 5853 calls 175
in lq 298 cost 787.166320850 cmp 137348 move 138347 calls 1
qs lq 397 cost 88.197632550 cmp 10509 move 6729 calls 195
in lq 397 cost 946.415952650 cmp 164912 move 165911 calls 1
qs lq 496 cost 89.009508850 cmp 11117 move 7755 calls 193
in lq 496 cost 1048.855918600 cmp 182643 move 183642 calls 1
numlq 6 limQuebras 1 lqdiff 607.914856

iter 1
qs lq 1 cost 74.492674050 cmp 6891 move 1969 calls 190
in lq 1 cost 4.183187050 cmp 1824 move 2823 calls 1
qs lq 40 cost 74.270163700 cmp 7102 move 2406 calls 190
in lq 40 cost 165.258493050 cmp 29704 move 30703 calls 1
qs lq 79 cost 89.759640600 cmp 8804 move 3186 calls 177
in lq 79 cost 307.632193400 cmp 54347 move 55346 calls 1
qs lq 118 cost 83.408099600 cmp 8694 move 3964 calls 174
in lq 118 cost 395.934739200 cmp 69631 move 70630 calls 1
qs lq 157 cost 83.987207050 cmp 8916 move 4299 calls 175
in lq 157 cost 510.478462900 cmp 89457 move 90456 calls 1
qs lq 196 cost 84.543155700 cmp 9200 move 4810 calls 196
in lq 196 cost 605.303749750 cmp 105870 move 106869 calls 1
numlq 6 limQuebras 1 lqdiff 303.449005

iter 2
qs lq 1 cost 74.492674050 cmp 6891 move 1969 calls 190
in lq 1 cost 4.183187050 cmp 1824 move 2823 calls 1
qs lq 16 cost 73.896124250 cmp 6927 move 2123 calls 187
in lq 16 cost 84.143095050 cmp 15664 move 16663 calls 1
qs lq 31 cost 75.610790250 cmp 7173 move 2323 calls 187
in lq 31 cost 142.379791050 cmp 25744 move 26743 calls 1
qs lq 46 cost 74.136264900 cmp 7172 move 2536 calls 181
in lq 46 cost 191.280127850 cmp 34208 move 35207 calls 1
qs lq 61 cost 86.762291400 cmp 8373 move 2802 calls 165
in lq 61 cost 264.255098800 cmp 46839 move 47838 calls 1
qs lq 76 cost 87.729320600 cmp 8595 move 3106 calls 177
in lq 76 cost 303.293328450 cmp 53596 move 54595 calls 1
numlq 6 limQuebras 16 lqdiff 138.196609
```

Figura 5: Exemplo de saída para calibração do número de quebras (parte 1)

```
iter 3
qs lq 1 cost 74.492674050 cmp 6891 move 1969 calls 190
in lq 1 cost 4.183187050 cmp 1824 move 2823 calls 1
qs lq 7 cost 73.739024400 cmp 6858 move 2008 calls 184
in lq 7 cost 33.376641900 cmp 6877 move 7876 calls 1
qs lq 13 cost 74.676518700 cmp 6999 move 2146 calls 190
in lq 13 cost 69.734134750 cmp 13170 move 14169 calls 1
qs lq 19 cost 73.527470100 cmp 6893 move 2116 calls 187
in lq 19 cost 106.709814750 cmp 19570 move 20569 calls 1
qs lq 25 cost 76.800496400 cmp 7239 move 2246 calls 181
in lq 25 cost 121.199659350 cmp 22078 move 23077 calls 1
qs lq 31 cost 75.610790250 cmp 7173 move 2323 calls 187
in lq 31 cost 142.379791050 cmp 25744 move 26743 calls 1
numlq 6 limQuebras 13 lqdiff 73.333176

iter 4
qs lq 7 cost 73.739024400 cmp 6858 move 2008 calls 184
in lq 7 cost 33.376641900 cmp 6877 move 7876 calls 1
qs lq 9 cost 73.662481800 cmp 6858 move 2020 calls 184
in lq 9 cost 46.739883750 cmp 9190 move 10189 calls 1
qs lq 11 cost 74.590557600 cmp 6951 move 2068 calls 190
in lq 11 cost 59.074739500 cmp 11325 move 12324 calls 1
qs lq 13 cost 74.676518700 cmp 6999 move 2146 calls 190
in lq 13 cost 69.734134750 cmp 13170 move 14169 calls 1
qs lq 15 cost 73.953531200 cmp 6927 move 2114 calls 187
in lq 15 cost 78.879838100 cmp 14753 move 15752 calls 1
qs lq 17 cost 73.876988600 cmp 6927 move 2126 calls 187
in lq 17 cost 95.074030450 cmp 17556 move 18555 calls 1
qs lq 19 cost 73.527470100 cmp 6893 move 2116 calls 187
in lq 19 cost 106.709814750 cmp 19570 move 20569 calls 1
numlq 6 limQuebras 15 lqdiff 25.339895

iter 5
qs lq 13 cost 74.676518700 cmp 6999 move 2146 calls 190
in lq 13 cost 69.734134750 cmp 13170 move 14169 calls 1
qs lq 14 cost 75.018328500 cmp 7026 move 2152 calls 193
in lq 14 cost 73.119720450 cmp 13756 move 14755 calls 1
qs lq 15 cost 73.953531200 cmp 6927 move 2114 calls 187
in lq 15 cost 78.879838100 cmp 14753 move 15752 calls 1
qs lq 16 cost 73.896124250 cmp 6927 move 2123 calls 187
in lq 16 cost 84.143095050 cmp 15664 move 16663 calls 1
qs lq 17 cost 73.876988600 cmp 6927 move 2126 calls 187
in lq 17 cost 95.074030450 cmp 17556 move 18555 calls 1
numlq 6 limQuebras 14 lqdiff 9.145703
```

Figura 6: Exemplo de saída para calibração do número de quebras (parte 2)

próxima, senão, realiza os experimentos para determinar os limiares de calibração (tamanho de partição e número de quebras).

5 Como será feita a entrega

5.1 Submissão

A entrega do TP1 compreende duas submissões:

VPL TP1: Submissão do código a ser submetido até **20/05/2025, 23:59** (o sistema vai ficar aberto madrugada adentro, mais para evitar problemas transientes de infraestrutura). **Não serão aceitas submissões em atraso.** Detalhes sobre a submissão do código são apresentados na Seção 5.3.

Relatório TP1: Arquivo PDF contendo a documentação do TP, assim como a avaliação experimental, conforme instruções, a ser submetido até **20/05/2025, 23:59** (o sistema vai ficar aberto madrugada adentro, mais para evitar problemas transientes de infraestrutura). **Não serão aceitas submissões em atraso.** 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[1], 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.
9. Número máximo de páginas incluindo a capa: 10

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

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:

```
– 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 - (15% 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 - (30% da nota total)
- Aderência completa às instruções de entrega - (5% da nota total)

Se o programa submetido **não compilar**[2] 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. 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.

[2]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.

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.