Documentação do trabalho prático
Programação e desenvolvimento de Software 2 - 2018/2
Nome: Igor Giovanelle Rezende Petrucci - 2015435632
Isaac Henrique Simões - 2015435713

1-Introdução:

O objetivo deste trabalho é realizar a modelagem de um conjunto de classes dotadas de métodos e atributos que permitam a implementação de um software para monitoramento de uma planta industrial de um processo de produção de leite condensado. O processo em questão possui um condensador, no qual ocorre todo o processo de transformação do leite em leite condensado. Para tal devem ser controladas as seguintes variáveis: Temperatura, Nível, Vazão, Pressão e Concentração. Para isso, é necessário a implementação da gestão de exceções para notificar situações anormais do sistema de produção.

2-Situação problema:

A máquina da planta industrial gera um arquivo de logs no qual informa o dia, horário e valores referentes a temperatura, nível, vazão, pressão e concentração:

2018/05/21 18:01:25 33.1;450.8;35.3;41.3;70;
2018/05/21 18:02:34 32.8;432.3;25.3;46.3;90;
2018/05/21 18:04:45 35.1;442.9;23.3;43.3;40;
Exemplo do arquivo de entrada

O programa desenvolvido para solução do problema tem como objetivo o tratamento das informações fornecidas pela máquina que possui um teclado numérico e monitoramento dos valores de restrição definidos pelo funcionário responsável, sendo assim, a restrição do sistema ocorre na definição dos limites superiores e inferiores, pois, caso um ou mais sensores sejam violados o programa deverá informar ao usuário o atributo e qual restrição foi desrespeitada. Além disso, é importante que o programa gere uma documentação de saída, no qual informa as restrições definidas, a priori, as informações de data, convertidas do padrão fornecido pela máquina para o formato PT-BR (dia, mês, ano), horário, e classificação de cada valor dos atributos tratados.

3-Desenvolvimento:

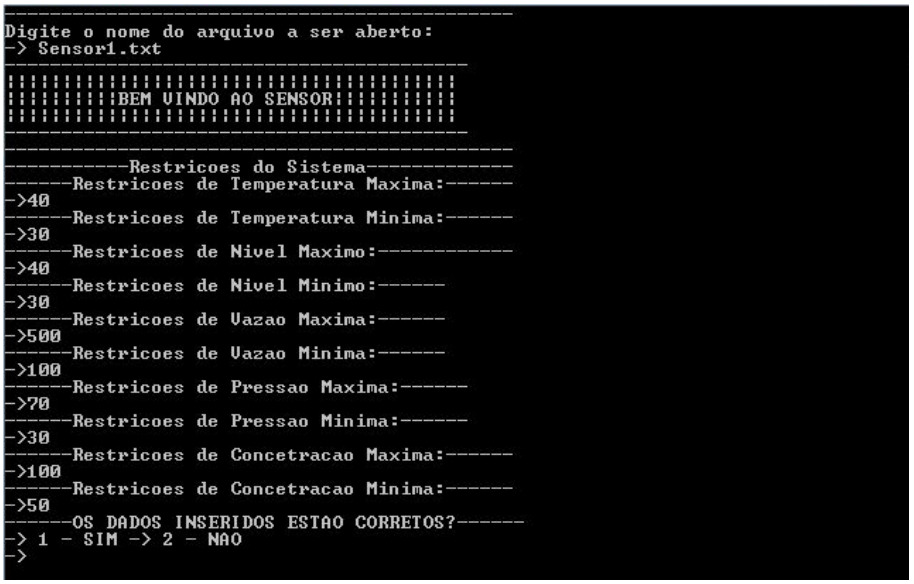
Foi utilizado o programa de diagrama de classes Astah no planejamento das classes Data, Hora, Sensor, Exceção, Abertura de Arquivo e Interface. O projeto tem seis etapas para a solução do problema, sendo eles: impressão da interface, abertura do arquivo, contagem de linhas do arquivo de entrada fornecido pela máquina, definição de restrições, tratamento e fechamento do arquivo.

Para este projeto foi desenvolvido a classe Data, pois, em C++, não há a classe padrão para o tratamento de dados referente. Nesta classe há a possibilidade de comparação de datas, saber quantos dias há entre elas, se a data é maior, menor e copiar dados de uma para outra.

Na classe de abertura de arquivo foram instanciadas variáveis das classes Data, Hora, Sensor e Exceção para a computação dos dados referente a cada linha do arquivo de entrada. Logo após a abertura do arquivo, o programa verifica a quantidade de logs criado pelo condensador. Cada linha do arquivo possui espaços e, a partir desses espaços, ocorre a separação de dados para o tratamento das informações.

A principal aplicação do projeto está no tratamento de exceção aos limites superiores e inferiores, pois a inserção de dados iguais a zero e, limite superior igual ao

limite inferior, pode ocorrer instabilidade do sistema e afetar seu desempenho. Quando identificado este tipo de erro, há o tratamento de dados e a reinserção de dados pelo usuário. O sistema não avança para o próximo item caso o usuário continue informando valores incompatíveis com o sistema. Caso deseje alterar um valor de restrição já definido, o usuário tem a opção de refazê-lo.



Exemplo do funcionamento da interface de definição dos limites superiores e inferiores

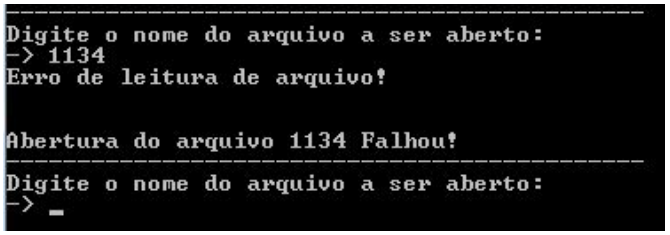
O tratamento de exceção precisa de uma variável booleana para permitir que o programa continue ou repita a ação para a inserção correta dos valores fornecidos pelo usuário. Para isso foi utilizado um do-while. O funcionamento dentro do try está relacionado com o valor digitado pelo usuário, neste exemplo o valor da concentração inferior não pode ser igual a zero ou ser igual à concentração superior, caso uma das situações ocorra o sistema avisa o usuário o erro e pede novo valor para a concentração.

```
//Variável booleana para controle do do while
      repeticao_ = false;

      do{
         try{
            cout << "------Restricoes de Concetracao Minima:------\n" << "->";
            cin >> concentracaoinferior_;
            if(concentracaoinferior_ <= .0 || concentracaoinferior_ >=
concentracaosuperior_){
               throw Excecao("-----CONCENTRACAO INFERIOR invalido!-----\n");
            }else{
               repeticao_ = true;
            }
         }catch(Excecao &e){
            e.Out();
            cout << "--Entre com um valor diferente de ZERO e menor que a
CONCENTRACAO SUPERIOR--\n" << endl;
         }
      }while(repeticao_ == false);
```

Exemplo da implementação do tratamento de exceção

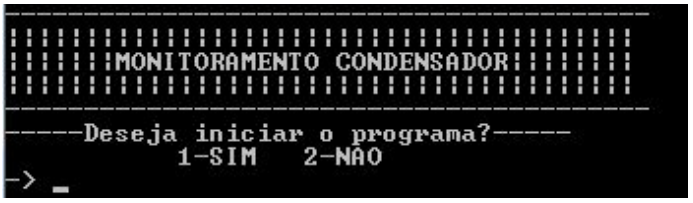
A abertura de arquivo é implementada da mesma forma que o controle dos limites superiores e inferiores, pois o sensor é identificado por um número onde os logs estão sendo armazenados, portanto, se o funcionário errar o nome do arquivo correspondente o sistema irá reiniciar e pedir novamente o nome correto.



Exemplo de leitura de arquivo

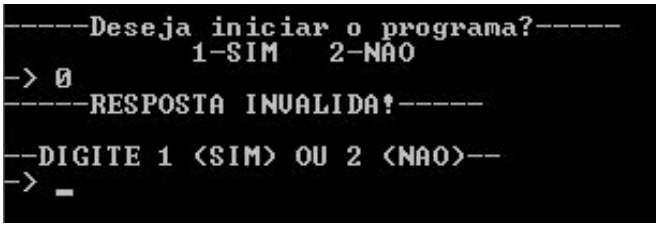
4-Interface:

A interface realiza o encapsulamento do sistema, portanto ao ser iniciado o programa pergunta ao usuário se deseja continuar a aplicação ou se deseja encerrar:



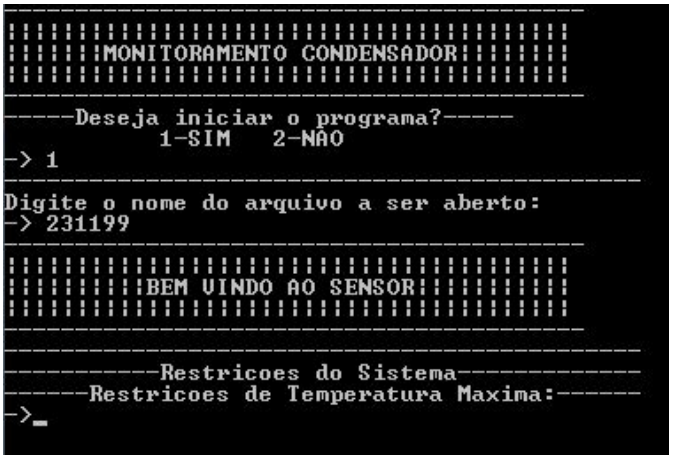
Exemplo do iniciação do programa

Caso o funcionário digite um valor diferente ao que foi indicado o programa realiza o tratamento de exceção, informa o erro “Resposta inválida” e pede que o usuário digite novamente uma escolha válida.



Exemplo de tratamento de exceção

Após a inicialização do programa e definição do arquivo entrada do condensador, o programa inicializa os valores referentes a restrição para monitoramento dos sensores. Como o usuário dispõem apenas de um teclado numérico, o sistema autocompleta a extensão do arquivo de texto referente ao nome do arquivo, por exemplo, em vez de digitar 231199.txt o sistema reconhece que 231199 já possui a extensão .txt.



Exemplo de funcionamento

O arquivo de saída conterá os dados referentes às restrições definida pelo usuário e os logs do condensador identificados com a data no formato PT-BR, horário, valores dos sensores nomeados e alertar emitidos pelos sistema.

```
------------231199.txt------------
------------RESTRICOES------------
Temperatura superior: 40
Temperatura inferior: 30
Nivel superior: 40
Nivel inferior: 30
Vazao superior: 500
Vazao inferior: 100
Pressao superior: 70
Pressao inferior: 30
Concentracao superior: 100
Concentracao inferior: 50
----------------------------------------
---------------LOGS--------------------
----------------------------------------
21/05/2018  18:01:25    Temperatura: 35.3  Nivel: 33.1  Vazao: 450.8  Pressao: 41.3  Concentracao: 70
21/05/2018  18:02:34    Temperatura: 25.3  Nivel: 32.8  Vazao: 432.3  Pressao: 46.3  Concentracao: 90
^^^^^^-------ALERTA DE TEMPERATURA MIN-------
21/05/2018  18:04:45    Temperatura: 23.3  Nivel: 35.1  Vazao: 442.9  Pressao: 43.3  Concentracao: 40
^^^^^^-------ALERTA DE TEMPERATURA MIN-------
-------ALERTA DE CONCENTRACAO MIN----------------------^^^^^^
21/05/2018    Temperatura: 35.3  Nivel: 33.1  Vazao: 450.8  Pressao: 41.3  Concentracao: 70
21/05/2018  18:11:34    Temperatura: 25.3  Nivel: 32.8  Vazao: 432.3  Pressao: 46.3  Concentracao: 90
^^^^^^-------ALERTA DE TEMPERATURA MIN-------
21/05/2018  18:32:45    Temperatura: 23.3  Nivel: 35.1  Vazao: 442.9  Pressao: 43.3  Concentracao: 40
^^^^^^-------ALERTA DE TEMPERATURA MIN-------
-------ALERTA DE CONCENTRACAO MIN----------------------^^^^^^
21/05/2018  18:01:25    Temperatura: 35.3  Nivel: 33.1  Vazao: 450.8  Pressao: 41.3  Concentracao: 70
21/05/2018  19:02:34    Temperatura: 25.3  Nivel: 32.8  Vazao: 432.3  Pressao: 46.3  Concentracao: 90
```

Ao término o programa perguntará ao usuário se deseja ler outro arquivo de sensor ou se deseja encerrar o programa.

```
------------------------------------------
------------FIM DE LEITURA----------------
------------------------------------------
------------------------------------------
------------------------------------------
---Deseja monitorar outro condensador?---
          1-SIM   2-NAO
->
```

5-Teste de unidade

Os testes de unidade foram realizados manualmente, pois ao instalar o Boot.teste e o Google test houve erro de compilação da biblioteca referente ao dois testes unitários, sendo assim, optamos por isolar cada trecho de código e testar separadamente a fim de identificar possíveis erros durante a execução do programa.

5.1 - Teste da classe Data

```
4   int main(){
5       //Interface t;
6
7   //Teste da CLASSE DATA
8       Data A;
9       A.SetDia(26);
10      A.SetMes(5);
11      A.SetAno(2016);
12
13      Data B;
14      B.SetDia(26);
15      B.SetMes(5);
16      B.SetAno(2018);
17      cout << "Dias: " << A.GetDia() << " Mes: " << A.GetMes() << " Ano: " << A.GetAno() << endl;
18      cout << "Total de dias eh: " << A.total_de_dias() << "\n\n" << endl;
19      cout << "Dias: " << A.GetDia() << " Mes: " << A.GetMes() << " Ano: " << B.GetAno() << endl;
20      cout << "Total de dias eh: " << B.total_de_dias() << endl;
21
```

Análise de dia, mês, ano com impressão na tela e a quantidade de dias a partir de 1 de janeiro de 1582.

```
Dias: 26 Mes: 5 Ano: 2016
Total de dias eh: 158624


Dias: 26 Mes: 5 Ano: 2018
Total de dias eh: 158921
```

Teste da classe data - impressão e quantidade de dias

Implementação para teste de sobrecarga de operador para saber se a data A é maior ou menor que B e se B é maior ou menor que A:

```
20  //    cout << "Total de dias eh: " << B.total_de_dias() << endl;
21
22      //Teste para saber se a data da esquerda é a menor
23      int bb = 9; //coloquei 9 apenas para que eu saiba se a var bb será 0 ou 1 de fato
24      bb = B<A;
25      cout << "\nB eh menor que A? -> " << bb << endl;   //Se o resultado der 1 (true) a sentença é verdadeira
26                          //Se o resultado der 0 (false) a sentença é falsa
27      int cc = 9;
28      cc = A<B;
29      cout << "A eh menor que B? -> " << cc << endl;
30
31      //Teste para saber se a data da esquerda é a maior
32      int dd = 9;
33      dd = B>A;
34      cout << "B eh maior que A? -> " << dd << endl;   //Se o resultado der 1 (true) a sentença é verdadeira
35                          //Se o resultado der 0 (false) a sentença é falsa
36      int ee = 9;
37      ee = A>B;
38      cout << "A eh maior que B? -> " << ee << endl;
39
```

```
A: Dias: 26 Mes: 5 Ano: 2016
B: Dias: 26 Mes: 5 Ano: 2018

B eh menor que A? -> 0
A eh menor que B? -> 1
B eh maior que A? -> 1
A eh maior que B? -> 0
```

Teste para analisar se A ou B são maiores (0 - False, 1 - True)

Teste da função que analisa a quantidade de dias entre duas datas, através da sobrecarga de operador -, neste caso de teste se refere a apenas um dia:

```
40      //Teste de subtração de datas para descobrir quantos dias há entre
41      int qnts_dias = 0;
42      qnts_dias = A-B;
43      cout << "Quantos Dias tem entre as datas: " << qnts_dias << endl;
44
```

```
Dia: 26 Mes: 5 Ano: 2016
Dia: 27 Mes: 5 Ano: 2016
Quantos Dias tem entre as datas: 1
```

Impressão na tela para saber quantos dias há entre

5.2- Teste da Classe Hora

Teste dos métodos Set e Get para alterações dos valores da classe Hora e impressão na tela dos atributos da Classe Hora dentro da função Imprime_Tempo().

```
15
16  void Hora::Imprime_Tempo(){
17      if(GetHora() < 10){
18          cout << " 0" << GetHora() << ":" << GetMinuto() << ":" << GetSegundo();
19      }
20      if(GetMinuto() < 10){
21          cout << " " << GetHora() << ":0" << GetMinuto() << ":" << GetSegundo();
22      }
23      if(GetSegundo() < 10){
24          cout << " " << GetHora() << ":" << GetMinuto() << ":0" << GetSegundo();
25      }
26      if((GetHora()<10) && (GetMinuto()<10) && (GetSegundo()<10) ){
27          cout << " 0" << GetHora() << ":0" << GetMinuto() << ":0" << GetSegundo() << endl;
28      }
29      //Se nenhum caso é atendido, imprime a hora
30      if((GetHora()>10) && (GetMinuto()>10) && (GetSegundo()>10) ){
31          cout << " " << GetHora() << ":" << GetMinuto() << ":" << GetSegundo() << endl;
32      }
33  }
34
```

A função Imprime_Tempo(), analisa se o número referente as horas, minutos e segundos se são menores que dez, pois o sistema não imprime o zero caso esteja a esquerda do número do Set, sendo assim, há a necessidade de completar a visualização

com zero para garantir a não ambiguidade de dados, por exemplo, impressão: 10:1:1, não se sabe se o 1 referente ao minuto ou segundo se é uma unidade ou se são dez.

5.3-Teste de abertura e fechamento de arquivo

```
4   int main(){
5       //Interface t;
6       Openarquivo teste_;
7       teste_.Abrearquivo();
8       //teste_.Contagemlinhas();
9       //teste_.Mostraarquivo();
10      //abre_.Tratamento();
11      teste_.Fechaaquivo();
12
```

```
C:\Users\captain9\Documents\PDS2\TESTE\bin\Debug\TESTE.exe
-----------------------------------------------
Digite o nome do arquivo a ser aberto:
-> Sensor1.txt

Abertura do arquivo Sensor1.txt realizada com sucesso!
-----------------------------------------------
---->Fechamento do arquivo realizado com sucesso!

Process returned 0 (0x0)   execution time : 3.679 s
Press any key to continue.
```

Teste de abertura e fechamento do arquivo

5.4-Teste de contagem de linhas

```
4   int main(){
5       //Interface t;
6       Openarquivo teste_;
7       teste_.Abrearquivo();
8       teste_.Contagemlinhas();
9       //teste_.Mostraarquivo();
10      //abre_.Tratamento();
11      teste_.Fechaaquivo();
```

```
-----------------------------------------------
Digite o nome do arquivo a ser aberto:
-> Sensor1.txt

Abertura do arquivo Sensor1.txt realizada com sucesso!
-----------------------------------------------
Quantidade de linhas eh: 3


---->Fechamento do arquivo realizado com sucesso!
```

Teste de contagem de linhas

5.5-Teste da função Tratamento()

Verifica se a função está separando as partes corretamente da linha do arquivo de entrada:

```
73  //Realiza a separação dos dados de cada linha e tratamento para alocação
74  //de memória para cada item, bem como a análise de sensibilidade se o sensor
75  //está identificando alguma situação de temperatura
76  void Openarquivo::Tratamento(){
77
78      string date_, hour_, value_;
79      for(int i=0; i < linhas_ ; i++){
80          leitura >> date_;
81          leitura >> hour_;
82          leitura >> value_;
83
84      cout << date_ << " " << hour_ << " " << value_ << endl;
85          ///Tratamento das informações
86          //Separadata(date_);
87          //Separahorario(hour_);
88          //Separavalores(value_);
89          //Analisavalores(); ///Verifica se os valores correspondem entre os limites superiores e inferiores
90      }
91  }
```

```
Digite o nome do arquivo a ser aberto:
-> Sensor1.txt

Abertura do arquivo Sensor1.txt realizada com sucesso!


Teste de leitura das linhas:
2018/05/21 18:01:25 33.1;450.8;35.3;41.3;70;
2018/05/21 18:02:34 32.8;432.3;25.3;46.3;90;
2018/05/21 18:04:45 35.1;442.9;23.3;43.3;40;
---->Fechamento do arquivo realizado com sucesso!
```

5.6 - Teste da função Separadata():

Isolando a função do restante do programa:

```
83
84          ///Tratamento das informações
85          Separadata(date_);
86          //Separahorario(hour_);
87          //Separavalores(value_);
88          //Analisavalores(); ///Verifica se os valores correspondem entre os limites superiores e inferiores
89      }
90  }
```

Implementação da função:

```
92  void Openarquivo::Separadata(string data_){
93      cout << "\nImprimindo data: " << data_ << "\n" << endl;
94      istringstream date(data_);
95      ///Separando em parcelas menores a string da data
96      string ano_, mes_, dia_;
97      getline(date, ano_, '/');
98      getline(date, mes_, '/');
99      getline(date, dia_, ' ');
100
101     int year_, month_, day_;
102     ///convertendo as parcelas da string em int
103     istringstream ano(ano_);
104     ano >> year_;
105     istringstream mes(mes_) ;
106     mes >> month_;
107     istringstream dia(dia_);
108     dia >> day_;
109
110     cout << "DATAS CONVERTIDAS: " << endl;
111     cout << "Dia: " << day_ << " Mes: " << month_ << " Ano: " << year_ << endl;
112
113     ///Definir na classe data os valores referentes ao ano, mes e dia
114     atual_.SetAno(year_);
115     atual_.SetMes(month_);
```

```
Digite o nome do arquivo a ser aberto:
-> Sensor1.txt

Abertura do arquivo Sensor1.txt realizada com sucesso!


Imprimindo data: 2018/05/21

DATAS CONVERTIDAS:
Dia: 21 Mes: 5 Ano: 2018

Imprimindo data: 2018/05/21

DATAS CONVERTIDAS:
Dia: 21 Mes: 5 Ano: 2018

Imprimindo data: 2018/05/21

DATAS CONVERTIDAS:
Dia: 21 Mes: 5 Ano: 2018
---->Fechamento do arquivo realizado com sucesso!
```

Impressão em tela da função de teste

5.7 - Teste da função Separahorario():

Isolar a função que separar horário para a análise:

```
83
84          ///Tratamento das informações
85          //Separadata(date_);
86          Separahorario(hour_);
87          //Separavalores(value_);
88          //Analisaparavaras(); ///Verifica se os valores correspondem entre os limites superiores e inferiores
89      }
90  }
```

Implementação da função:

```
116
117 void Openarquivo::Separahorario(string data_){
118     cout << "\n\nImprimido horario: " << data_ << endl;
119
120     istringstream tempo(data_);
121
122     string hora_, minuto_, segundo_;
123     getline(tempo,hora_,':');
124     getline(tempo,minuto_,':');
125     getline(tempo,segundo_,' ');
126
127     int hour_,minute_,second_;
128     istringstream hora(hora_);
129     istringstream minuto(minuto_);
130     istringstream segundo(segundo_);
131     hora >> hour_;
132     minuto >> minute_;
133     segundo >> second_;
134
135     cout << "\nHORA CONVERTIDA:" << endl;
136     cout << " Hora: " << hour_ << " Minuto: " << minute_ << " Segundos: " << second_ << endl;
137
138
```

```
Digite o nome do arquivo a ser aberto:
-> Sensor1.txt

Abertura do arquivo Sensor1.txt realizada com sucesso!
----------------------------------------------


Imprimido horario: 18:01:25

HORA CONVERTIDA:
 Hora: 18 Minuto: 1 Segundos: 25


Imprimido horario: 18:02:34

HORA CONVERTIDA:
 Hora: 18 Minuto: 2 Segundos: 34


Imprimido horario: 18:04:45

HORA CONVERTIDA:
 Hora: 18 Minuto: 4 Segundos: 45
---->Fechamento do arquivo realizado com sucesso!
```

Impressão da função de teste

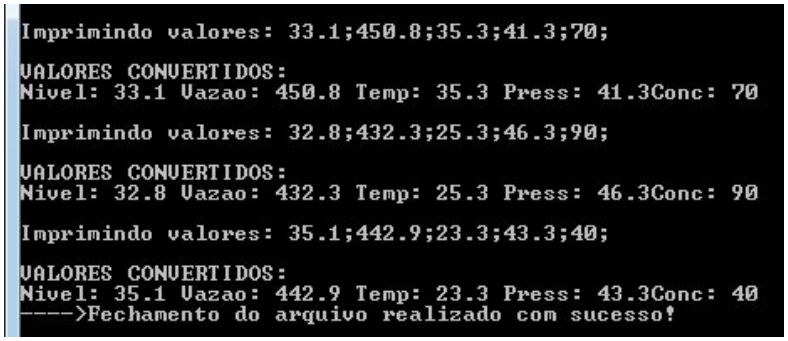
5.8 - Teste da função Separavaores():

Isolando a função do restante do programa:

```
84          ///Tratamento das informações
85          //Separadata(date_);
86          //Separahorario(hour_);
87          Separavalores(value_);
88          //Analisaparavaras(); ///Verifica se os valores correspondem entre os limites superiores e inferiores
89      }
90  }
```

Implementação da função:

```
146     string nivel_, vazao_, temperatura_, pressao_, concentracao_;
147     getline(sensor,nivel_,';');
148     getline(sensor,vazao_,';');
149     getline(sensor,temperatura_,';');
150     getline(sensor,pressao_,';');
151     getline(sensor,concentracao_,' ');
152
153     double dnivel_, dvazao_, dtemperatura_, dpressao_, dconcentracao_;
154     istringstream nivel(nivel_);
155     istringstream vazao(vazao_);
156     istringstream temperatura(temperatura_);
157     istringstream pressao(pressao_);
158     istringstream concentracao(concentracao_);
159
160     nivel >> dnivel_;
161     vazao >> dvazao_;
162     temperatura >> dtemperatura_;
163     pressao >> dpressao_;
164     concentracao >> dconcentracao_;
165
166     cout << "\nVALORES CONVERTIDOS: " << endl;
167     cout << "Nivel: " << dnivel_ << " Vazao: " << dvazao_ << " Temp: " << dtemperatura_
168          << " Press: " << dpressao_ << "Conc: " << dconcentracao_ << endl;
```



Impressão em tela do teste da função

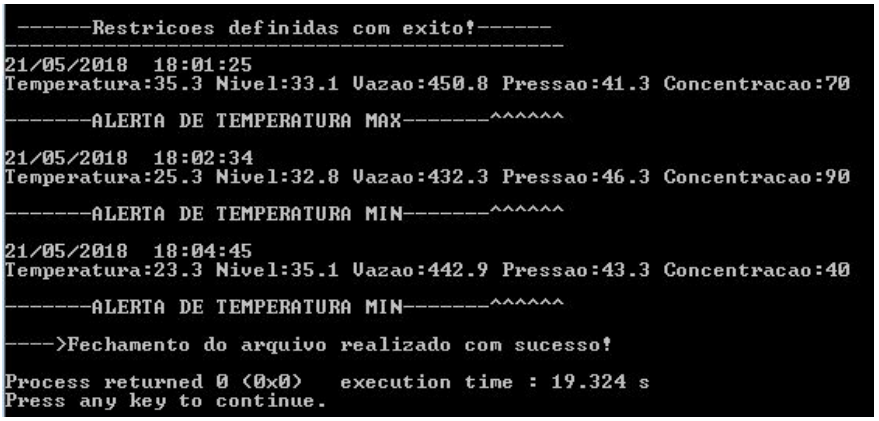
5.9- Teste das restrições na função Analisavalores():

Isolar a função dentro do if para teste de dados:

```
196     //Grava os valores analisados no arquivo base
197     Gravalogs();
198         if(dados_.Gettemperatura() > restricoes_.Gettempsuperior()){
199             Excecao e("-------ALERTA DE TEMPERATURA MAX--------^^^^^^\n");
200             Gravalogs(e.Geterro());
201             e.Out();
202         }
```

Trecho de código utilizado

A utilização de if's em vez do Try-Catch se deve a possibilidade de haver mais de uma violação de restrição. Quando o programa identifica a exceção, a função Throw para a execução da linha de código e realiza o tratamento, porém, o programa não verifica se existe outras violações do sistema a partir da linha interrompida, optamos em utilizar o if em relação ao Try-Catch neste trecho. A informação perdida no tratamento é importante para o usuário.



Teste de função para restrição de Temperatura máxima e mínima

7- Git e GitHub

Utilizamos o Git e GitHub para controle de versão do programa. Anteriormente era realizado o backup manual dos arquivos via nuvem, após a aula sobre controle de versão passamos a utilizar no workflow o repositório online para armazenar o progresso do trabalho.

Endereço onde os commits estão salvos na branch master:
https://github.com/igorpetrucci/TP_PDS2_20182
https://github.com/caasihenrique/Tp_Final_PDS2

7-Conclusões finais

A linguagem orientada a objetos permite inúmeras possibilidades e abre caminho para novas aplicações, é de extrema importância o seu aprendizado para os engenheiros, pois, com a disciplina de programação e desenvolvimento de software conseguimos entender a visão de projeto de grandes proporções e aplicar a programação defensiva de erros de sistema, além disso, aplicá-lo de forma prática em atividades laboratoriais.