

## **Documentação do trabalho prático**

## **Programação e desenvolvimento de Software 2 - 2018/2**

Nomes: 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 e controle 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;
- Concentração;

Além disso, há a implementação da gestão de exceções para indicar uma situação anormal de produção. Sabe-se que o tempo em que o leite fica no condensador deve ser controlado, assim como a temperatura e pressão na qual fica submetido durante o processo. O funcionário irá inserir no programa de exceções às restrições iniciais do limite superior e inferior de cada variável.

### **2- Planejamento**

A análise de como é fabricado o leite condensado é de extrema importância para o desenvolvimento para o programa, pois consiste na centrifugação, filtração, pasteurização, adição de açúcar, vaporização e adição de lactose. Todas as etapas requerem atenção aos atributos de temperatura, pressão e concentração, uma vez que qualquer oscilação extrema pode acarretar na perda da produção de leite condensado. Portanto, o planejamento do programa de monitoramento se deve a análise prévia do comportamento do condensador e seu mapeamento através das classes a serem implementadas, para isso, utilizamos o programa Astah.

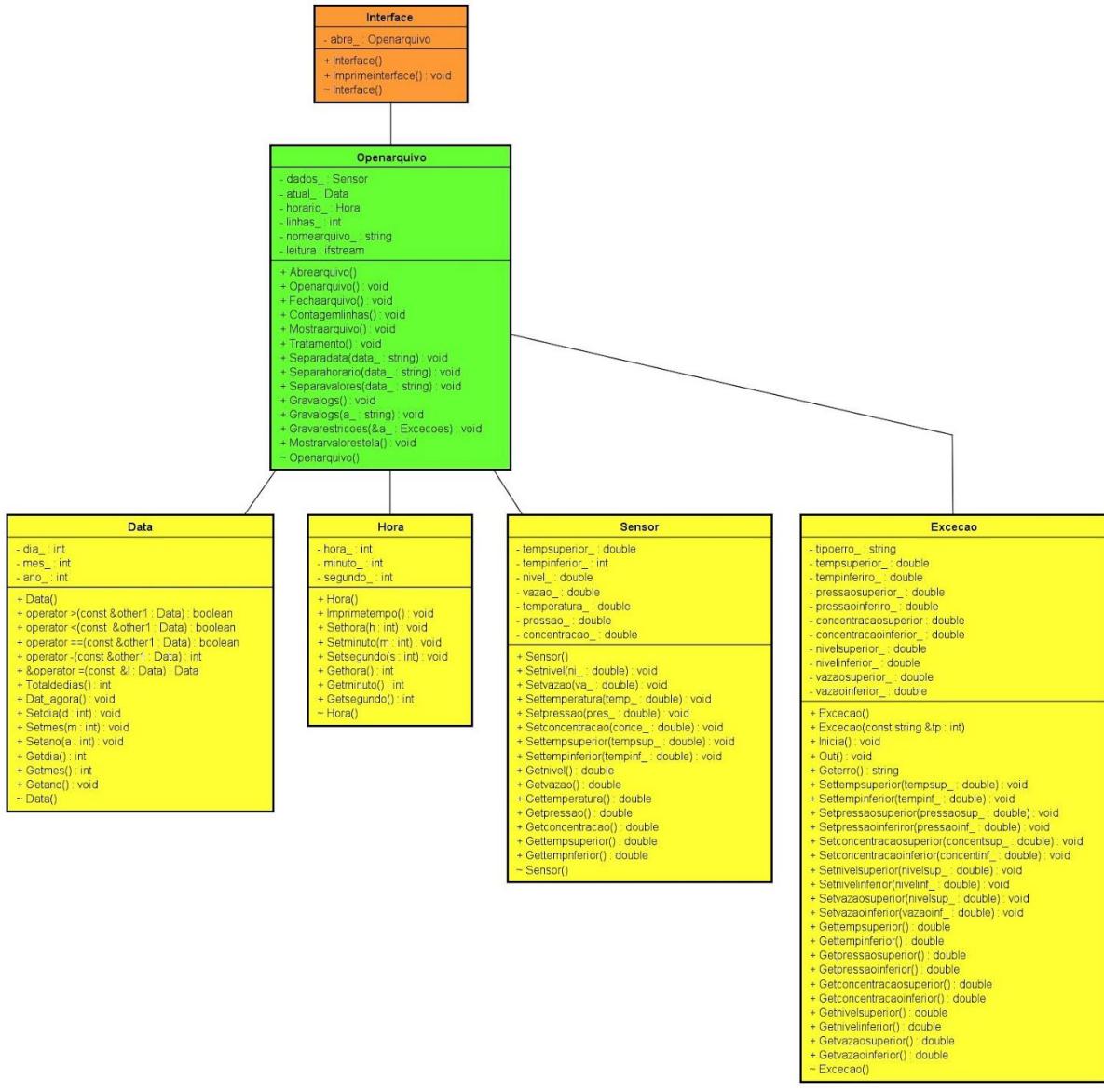
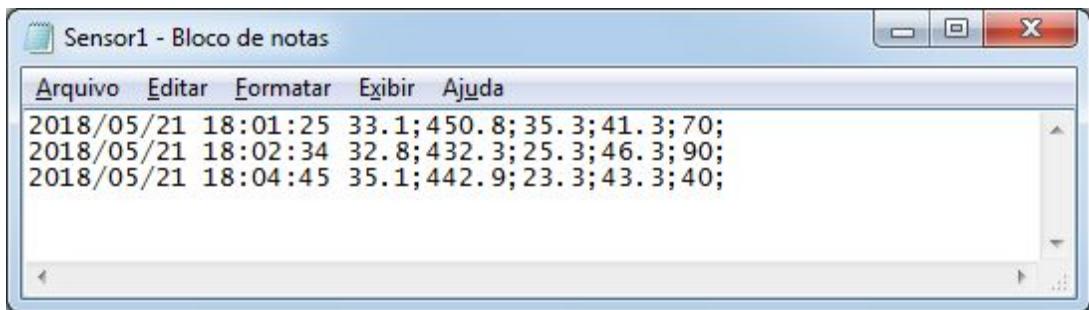


Diagrama de classes para o programa de monitoramento  
 (Imagem disponível na pasta do projeto)

A construção das classes partiu do nível mais baixo (data, hora, sensor e exceção) para o nível mais alto (Interface).

O programa irá ler de um arquivo, no formato “.txt”, logs de máquina com dados referente ao horário e valores de temperatura, nível do tanque, vazão, pressão e concentração para o monitoramento. Os dados estão listados linha por linha na seguinte ordem: data, horário, nível, vazão, temperatura, pressão, concentração.



Modelo de arquivo .txt para leitura dos dados

A premissa do programa é identificar cada log e analisar os limites superiores e inferiores informados pelo usuário do sistema, por exemplo, se houve superaquecimento ou se a altura máxima ou mínima do reservatório foi atingida. Após identificação é dever do programa identificar e catalogar em um arquivo de saída, no formato .txt, os logs referentes a violação das restrições.

## 2.1 - Manual de estilo adotado

O manual de estilo adotado refere-se ao mencionado pelo professor durante as aulas da disciplina Programação e Desenvolvimento de Software 2, nos quais todas as variáveis possuem letras minúsculas acompanhado do underscore (underline ou sublinhado), por exemplo: **variavel\_**, desse modo é possível identificar quais atributos estamos tratando e diferenciá-los de funções a serem chamadas. As funções e Classes possuem a primeira letra maiúscula, por exemplo: **Sethorariostring()**.

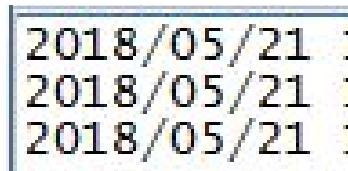
```
15     public:  
16         Openarquivo();  
17         void Abrearchivo();  
18         void Fechaaquivo();  
19         void Contagemlinhas();  
20         void Mostraarquivo();  
21         void Tratamento();  
22         void Separadata(string data_);  
23         void Separahorario(string data_);  
24         void Separavalores(string data_);  
25         void Analisavalores();  
26         void Gravalogs();  
27         void Gravalogs(string a_);  
28         void Gravarestricoes(Excecao &restricoes_);  
29         void Mostravalorestela();  
30         ~Openarquivo();  
31  
32     private:  
33         Sensor dados_;  
34         Data atual_;  
35         Hora horario_;  
36         bool rest_;  
37         int linhas_;
```

Exemplo do manual de estilo adotado na Classe Openarquivo

Quanto a utilização de nomes em inglês e português, não houve restrição na decisão da equipe.

## 3- Implementação

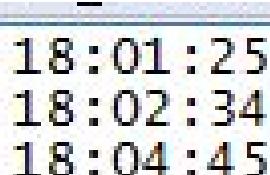
Como a biblioteca padrão C++ não possui uma classe que trata especificamente data, foi necessário o desenvolvimento da classe que trate a data, pois o formato fornecido pela máquina é diferente ao utilizado no Brasil, sendo assim necessário converter o formato ano-mês-dia para dia-mês-ano para que haja melhor assimilação do usuário quanto ao formato utilizado no programa.



```
2018/05/21
2018/05/21
2018/05/21
```

Formato de data fornecido pelo log de máquina

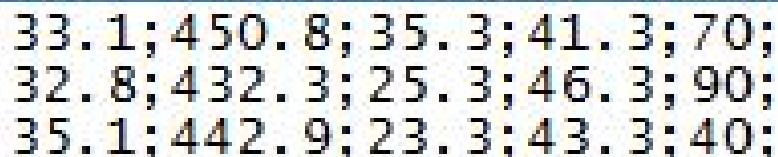
Após a identificação da data e armazenamento dos dados na classe Data, é necessário a criação da classe Hora para armazenamento e tratamento das informações da hora, minuto e segundo:



```
—
18:01:25
18:02:34
18:04:45
```

Formato horário fornecida pelo log de máquina

Por último há a leitura dos dados referentes às variáveis da classe Sensor, onde estão os atributos da temperatura, nível, vazão, pressão, concentração:



```
33.1;450.8;35.3;41.3;70;
32.8;432.3;25.3;46.3;90;
35.1;442.9;23.3;43.3;40;
```

Formato dos dados dos sensores de temperatura

### **3.1- Classe Data:**

O desenvolvimento da classe data tem como principal função a análise da quantidade de dias, verificar se as datas são iguais e qual a mais distante. Para isso são implementados sobrecargas de operador para facilitar a manipulação da classe.

```

6   using std::string;
7
8   class Data{
9     public:
10    Data();
11    //Sobrecarga de Operador
12    bool operator> (const Data &other1);
13    bool operator< (const Data &other1);
14    bool operator==(const Data &other1);
15    int operator-(Data &other1);
16    Data & operator=(const Data &l);
17
18    //Métodos adjacentes para sobrecarga de operador
19    int total_de_dias() const;
20    void dat_agora();
21    string Datastring();
22
23    void SetDia(int d);
24    void SetMes(int m);
25    void SetAno(int a);
26    int GetDia() const;
27    int GetMes() const;
28    int GetAno() const;
29
30    ~Data();
31
32  private:
33    int dia;
34    int mes;
35    int ano;

```

A função Datastring realiza a conversão dos valores int dos atributos dia, mês e ano para o tipo string, desse modo, podemos retornar o formato PT-BR referente a padronização do sistema. Foi utilizado a função que converte int em string pela biblioteca sstream através da stringstream:

```

123
124  string Data::Datastring(){
125    string data_;
126    stringstream dia_, mes_, ano_;
127
128    dia_ << dia;
129    mes_ << mes;
130    ano_ << ano;
131
132    if(dia < 10 && mes < 10){
133      data_ = "0" + dia_.str() + "/0" + mes_.str() + "/" + ano_.str() + " ";
134    }
135    if(dia < 10 && mes > 10){
136      data_ = "0" + dia_.str() + "/" + mes_.str() + "/" + ano_.str() + " ";
137    }
138    if(dia > 10 && mes > 10){
139      data_ = dia_.str() + "/" + mes_.str() + "/" + ano_.str() + " ";
140    }
141    if(dia > 10 && mes < 10){
142      data_ = dia_.str() + "/0" + mes_.str() + "/" + ano_.str() + " ";
143    }
144
145  }

```

### 3.2- Classe Hora:

Tem como funcionalidade o armazenamento dos atributos hora, minuto e segundo.

```

5
6     using std::string;
7
8     class Hora{
9         private:
10            int hora;
11            int minuto;
12            int segundo;
13
14        public:
15            Hora();
16            void Imprime_Tempo();
17            int GetHora() const;
18            int GetMinuto() const;
19            int GetSegundo() const;
20            void SetHora(int h);
21            void SetMinuto(int m);
22            void SetSegundo(int s);
23            string Horastring();
24
25            ~Hora();
26
27     };

```

Há a função Horastring que retorna uma string para exibição em tela e gravação no arquivo de saída, para isso ocorre a conversão dos valores int para o formato de string utilizado pela biblioteca stringstream através da stringstream:

```

56     string Hora::Horastring(){
57         string horario_;
58         stringstream hora_, minuto_, segundo_;
59
60         hora_ << hora;
61         minuto_ << minuto;
62         segundo_ << segundo;
63
64         if(minuto < 10 && segundo < 10){
65             horario_ = hora_.str() + ":0" + minuto_.str() + "0" + segundo_.str() + " ";
66         }
67         if(minuto < 10 && segundo > 10){
68             horario_ = hora_.str() + ":0" + minuto_.str() + ":" + segundo_.str() + " ";
69         }
70         if(minuto > 10 && segundo < 10){
71             horario_ = hora_.str() + ":" + minuto_.str() + "0" + segundo_.str() + " ";
72         }
73         if(minuto > 10 && segundo > 10){
74             horario_ = hora_.str() + ":" + minuto_.str() + ":" + segundo_.str() + " ";
75         }
76
77         return horario_;
78     }

```

### 3.3- Classe Sensor:

Atributos e métodos para monitoramento:

```

4
5 class Sensor{
6     public:
7         Sensor();
8         void Setnivel(double ni_);
9         void Setvazao(double va_);
10        void Settemperatura(double temp_);
11        void Setpressao(double pres_);
12        void Setconcentracao(double conce_);
13        double Getnivel();
14        double Getvazao();
15        double Gettemperatura();
16        double Getpressao();
17        double Getconcentracao();
18        ~Sensor();
19
20    private:
21        double nivel_;
22        double vazao_;
23        double temperatura_;
24        double pressao_;
25        double concentracao_;
26    };

```

### 3.4- Classe Exceção

A classe Exceção é o armazenamento dos limites superiores e inferiores de cada sensor do sistema. Quando ocorre a violação de uma restrição o try-catch é ativado e envia uma mensagem de qual tipo de exceção ocorreu, desse modo o sistema pode realizar a gravação do log no arquivo de saída e exibição na tela para informar ao usuário o tipo de exceção..

```

12 class Excecao{
13     public:
14         Excecao();
15         Excecao(const string &tp);
16         void Inicia();
17         void Out();
18         string Geterro();
19         void Settempsuperior(double tempsup_);
20         void Settempinferior(double tempinf_);
21         void Setpressaosuperior(double pressaosup_);
22         void Setpressaoinferior(double pressaoinf_);
23         void Setconcentracaosuperior(double concentsup_);
24         void Setconcentracaoinferior(double concentinf_);
25         void Setniveissuperior(double nivelsup_);
26         void Setnivelinferior(double nivelinf_);
27         void Setvazaosuperior(double vazaosup_);
28         void Setvazaoinferior(double vazaoinf_);
29         double Gettempsuperior();
30         double Gettempinferior();
31         double Getpressaosuperior();
32         double Getpressaoinferior();
33         double Getconcentracaosuperior();
34         double Getconcentracaoinferior();
35         double Getniveisuperior();
36         double Getnivelinferior();
37         double Getvazaosuperior();
38         double Getvazaoinferior();
39
40     ~Excecao();

```

```

41     private:
42         string tipoerro_;
43         double tempsuperior_;
44         double tempinferior_;
45         double pressaosuperior_;
46         double pressaoinferior_;
47         double concentracaosuperior_;
48         double concentracaoinferior_;
49         double niveisuperior_;
50         double nivelinferior_;
51         double vazaosuperior_;
52         double vazaoinferior_;
53     };
54
55

```

#### 4-Implementação classe Openarquivo

Esta classe é responsável pela abertura do arquivo, leitura dos dados, tratamento das informações e análise dos valores da linha do arquivo de entrada.

```

14 class Openarquivo{
15     public:
16         Openarquivo();
17         void Abre arquivo();
18         void Fecha arquivo();
19         void Contagemlinhas();
20         void Mostra arquivo();
21         void Tratamento();
22         void Separadata(string data_);
23         void Separahorario(string data_);
24         void Separavalores(string data_);
25         void Analisavalores();
26         void Gravatalogs();
27         void Gravatalogs(string a_);
28         void Gravarestricoes(Excecao &restricoes_);
29         void Mostrarvalorestela();
30         void Definerestricoes();
31         ~Openarquivo();
32
33     private:
34         Sensor dados_;
35         Data atual_;
36         Hora horario_;
37         Excecao restricoes_;
38         int linhas_;
39         string nomearquivo_;
40         ifstream leitura;
41     };
42

```

Cada função da classe Openarquivo() está representada abaixo, juntamente com sua implementação:

```
void Abre arquivo();
```

O funcionamento da função Abre arquivo() é obter através do usuário, o nome do arquivo referente ao condensador a ser analisado:

```

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

Abertura do arquivo Sensor1.txt realizada com sucesso!

```

Para a implementação da abertura de arquivo, utilizamos o tratamento de exceção, pois caso o usuário digite o nome errado do arquivo não seja preciso reiniciar o programa para uma nova inserção:

```
Digite o nome do arquivo a ser aberto:  
-> Sensor1.ttx  
Erro de leitura de arquivo!
```

```
Abertura do arquivo Sensor1.ttx Falhou!
```

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

Exibição caso ocorra o erro

```
22 void Openarquivo::Abrearquivo() {  
23     bool repeticao_ = true;  
24     do{  
25         try{  
26             cout << "-----" << endl;  
27             cout << "Digite o nome do arquivo a ser aberto: \n" << "-> ";  
28             cin >> nomearquivo_;  
29  
30             leitura.open(nomearquivo_.c_str());  
31             if (leitura.fail()){  
32                 throw Excecao("Erro de leitura de arquivo! \n");  
33             }else{  
34                 repeticao_ = true;  
35             }  
36         }catch(Excecao &e){  
37             e.Out();  
38             cout << "\nAbertura do arquivo " << nomearquivo_ << " Falhou!" << endl;  
39             repeticao_ = false;  
40         }  
41     }while(repeticao_ == false);  
42 }  
43  
44 }
```

Implementação da função Abrearquivo()

```
void Contagemlinhas();
```

A função Contagemlinhas() realiza a verificação de quantas linhas de log o arquivo de entrada possui, delimitar o tamanho do arquivo é de extrema importância, pois evita que o programa acesse áreas não permitidas ao ponteiro de leitura.

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

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

```
Quantidade de linhas eh: 3
```

```
void Mostraarquivo();
```

Realiza o teste de leitura e impressão dos dados referente a cada linha lida. Esta função tem como objetivo a verificação de dados e como estão sendo capturados pela string.

```
Digite o nome do arquivo a ser aberto:  
-> Sensor1.txt  
  
Abertura do arquivo Sensor1.txt realizada com sucesso!  
  
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!  
  
Process returned 0 <0x0> execution time : 4.450 s  
Press any key to continue.
```

Impressão em tela das lidas lida do arquivo modelo do condensador

```

59
60     void Openarquivo::Mostraarquivo(){
61         //Função para visualização da tela para usuário
62         string parte1_, parte2_, parte3_;
63         // cout << "Quantidade de linhas eh: << linhas_ << endl;
64         for(int i=0; i < linhas_; i++){
65             leitura >> parte1_;
66             leitura >> parte2_;
67             leitura >> parte3_;
68             cout << parte1_ << " " << parte2_ << " " << parte3_ << endl;
69         }
70     }

```

A função tem como funcionamento a quebra da linha em três partes, para poder verificar se há algum erro de leitura das informações, já que cada trecho é separado por espaços e uma quebra de linha \n.

```
void Definerestricoes();
```

Esta função realiza definição das restrições dos limites superiores e inferiores da classe Excecao:

```

312
313     void Openarquivo::Definerestricoes(){
314
315         //Realiza a gravação de das restrições nos atributos
316         restricoes_.Inicia();
317         Gravarestricoes(restricoes_);
318
319         string cabecalho_ = "-----\n-----LOGS-----\n";
320
321         Gravalogs(cabecalho_);
322
323         cabecalho_ = "-----\n";
324         Gravalogs(cabecalho_);
325
326     }

```

Início do processo de inicialização de restrições. Esta função tem como principal objetivo o tratamento das informações digitadas pelo usuário e definição de seus limites:

```

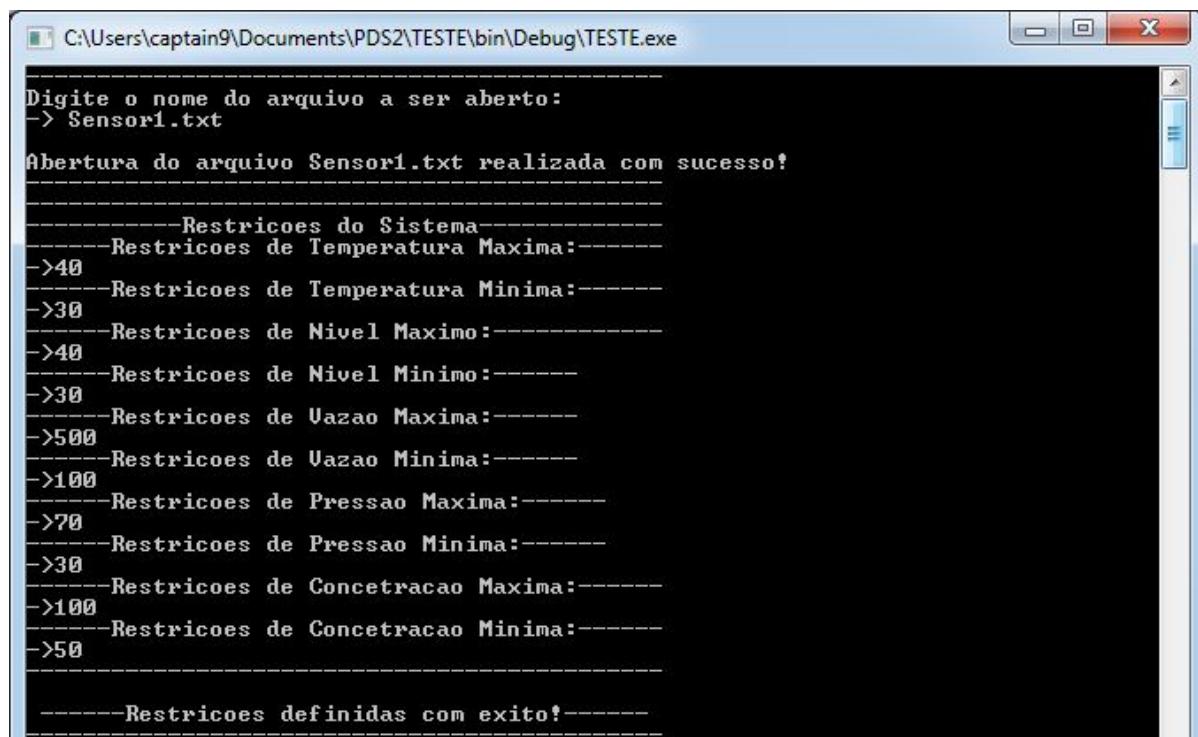
7     void Excecao::Inicia(){
8         tempsuperior_ = 0.;
9         tempinferior_ = 0.;
10        nivelsuperior_ = 0.;
11        nivelinferior_ = 0.;
12        vazaosuperior_ = 0.;
13        vazaoinferior_ = 0.;
14        pressaosuperior_ = 0.;
15        concentracaosuperior_ = 0.;
16        concentracaoinferior_ = 0.;

17        cout << "-----" << endl;
18        cout << "-----Restricoes do Sistema-----" << endl;
19        bool repeticao_ = false;
20        do{
21            try{
22                cout << "-----Restricoes de Temperatura Maxima:-----\n" << "->";
23                cin >> tempsuperior_;
24                if(tempsuperior_ <= .0){
25                    throw Excecao("-----TEMPERURA SUPERIOR invalido!-----\n");
26                }else{
27                    repeticao_ = true;
28                }
29            }catch(Excecao &e){
30                e.Out();
31                cout << "-----Entre com um valor maior que ZERO para Temperatura Maxima-----" << endl;
32            }
33        }while(repeticao_ == false);

34        //volta a variavel repeticao para o valor inicial
35        repeticao_ = false;
36
37    }

```

Após a definição dos valores é realizado a gravação das restrições no arquivo de saída.



```
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!
-----Restricoes do Sistema-----
Restricoes de Temperatura Maxima:-----
>40
Restricoes de Temperatura Minima:-----
>30
Restricoes de Nivel Maximo:-----
>40
Restricoes de Nivel Minimo:-----
>30
Restricoes de Vazao Maxima:-----
>500
Restricoes de Vazao Minima:-----
>100
Restricoes de Pressao Maxima:-----
>70
Restricoes de Pressao Minima:-----
>30
Restricoes de Concentracao Maxima:-----
>100
Restricoes de Concentracao Minima:-----
>50
-----Restricoes definidas com exito!-----
```

Impressão em tela das restrições definidas pelo usuário

```
-----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
```

Gravação do arquivo de saída com os valores definidos

---

```
void Tratamento();
```

Como a função MostrarValores(), a função tratamento realiza a quebra da linha em três partes sendo, data, hora e valores do sensor, assim, podemos tratar separadamente os atributos da classe Hora, Data e Sensor:

```

71 //Realiza a separação dos dados de cada linha e tratamento para alocação
72 //de memória para cada item, bem como a análise de resultado se o sensor
73 //já identificar alguma alteração de temperatura
74
75 void Openarquivo::Tratamento(){
76
77     string date_, hour_, value_;
78
79     for(int i=0; i < linhas_ ; i++){
80         leitura >> date_;
81         leitura >> hour_;
82         leitura >> value_;
83
84
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 }
92

```

Para tratamento das informações é necessário a chamada da função Separadata(string data\_)

```
void Separadata(string data_);
```

Ao quebrar os dados geramos uma string que será recebida pela função. O principal objetivo é separar cada trecho delimitado por '/' e transformar a string em int, logo, para isso é preciso utilizar a variável istringstream que realiza a conversão do ano, mês e dia (referente ao padrão da máquina) para seu respectivo atributo na classe Hora(Dia, Mês e Ano):

```

92
93     void Openarquivo::Separadata(string data_){
94         //cout << data_ << endl;
95         istringstream date(data_);
96         //Separando em parcelas menores a string da data
97         string ano_, mes_, dia_;
98         getline(date, ano_, '/');
99         getline(date, mes_, '/');
100        getline(date, dia_, ' ');
101
102        int year_, month_, day_;
103        //Convertendo as parcelas da string em int
104        istringstream ano(ano_);
105        ano >> year_;
106        istringstream mes(mes_);
107        mes >> month_;
108        istringstream dia(dia_);
109        dia >> day_;
110
111        //Definir na classe data os valores referentes ao ano, mes e dia
112        atual_.SetAno(year_);
113        atual_.SetMes(month_);
114        atual_.SetDia(day_);
115    }
116
void Separahorario(string data_);
```

O mesmo algoritmo funciona para a função Separahorario(string data\_):

```

116
117 void Openarquivo::Separahorario(string data_){
118     //cout << 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     //Definir na classe Hora os valores referentes ao ano, mes e dia
136     horario_.SetHora(hour_);
137     horario_.SetMinuto(minute_);
138     horario_.SetSegundo(second_);
139 }
140
void Separavalores(string data_);

```

mesmo procedimento para a função Separavalores(string data\_):

```

main.cpp X Openarquivo.cpp X Interface.cpp X Interface.h X
142 void Openarquivo::Separavalores(string data_){
143     //cout << data_ << endl;
144     istringstream sensor(data_);
145
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     //Definir na classe Dados os valores referentes ao ano, mes e dia
167     dados_.Setnivel(dnivel_);
168     dados_.Setvazao(dvazao_);
169     dados_.Settemperatura(dtemperatura_);
170     dados_.Setpressao(dpressao_);
171     dados_.Setconcentracao(dconcentracao_);
172

```

---

```
void AnalisaValores();
```

```

174 void Openarquivo::Analisavalores() {
175
176     //Tira os totais os valores lidos do arquivo de entrada
177     Mostrarvalorestela();
178
179     //Grava os valores analisados no arquivo base
180     Gravalogs();
181     if(dados_.Gettemperatura() > restricoes_.Gettempsuperior()){
182         Excecao e("-----ALERTA DE TEMPERATURA MAX-----^^^^^\\n");
183         Gravalogs(e.Geterro());
184         e.Out();
185     }
186     if(dados_.Gettemperatura() < restricoes_.Gettempinferior()){
187         Excecao e("-----ALERTA DE TEMPERATURA MIN-----^^^^^\\n");
188         Gravalogs(e.Geterro());
189         e.Out();
190     }
191     if(dados_.Getnivel() > restricoes_.Getnivalsuperior()){
192         Excecao e("-----ALERTA DE NIVEL MAX-----^^^^^\\n");
193         Gravalogs(e.Geterro());
194         e.Out();
195     }
196     if(dados_.Getnivel() < restricoes_.Getnivelinferior()){
197         Excecao e("-----ALERTA DE NIVEL MIN-----^^^^^\\n");
198         Gravalogs(e.Geterro());

```

Após a definição das restrições o sistema analisa se há infrações dos limites estabelecidos pelo usuário:

```

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     }
203     if(dados_.Gettemperatura() < restricoes_.Gettempinferior()){
204         Excecao e("-----ALERTA DE TEMPERATURA MIN-----^^^^^\\n");
205         Gravalogs(e.Geterro());
206         e.Out();
207     }
208     if(dados_.Getnivel() > restricoes_.Getnivalsuperior()){
209         Excecao e("-----ALERTA DE NIVEL MAX-----^^^^^\\n");
210         Gravalogs(e.Geterro());
211         e.Out();
212     }
213     if(dados_.Getnivel() < restricoes_.Getnivelinferior()){
214         Excecao e("-----ALERTA DE NIVEL MIN-----^^^^^\\n");
215         Gravalogs(e.Geterro());
216         e.Out();
217     }
218     if(dados_.Getvazaos() > restricoes_.Getvazaosuperior()){
219         Excecao e("-----ALERTA DE VAZAO MAX-----^^^^^\\n");
220         Gravalogs(e.Geterro());

```

Antes de cada if o sistema grava a linha referente a leitura do arquivo, através da função Gravalogs():

```

250 //Grava os logs lidos do sistema para o arquivo de saída
251 void Openarquivo::Gravalogs(){
252     ofstream Hypnos_FILE;
253
254     stringstream temp_, nivel_, vazao_, pressao_, concentracao_;
255
256     temp_ << dados_.Gettemperatura();
257     nivel_ << dados_.Getnivel();
258     vazao_ << dados_.Getvazao();
259     pressao_ << dados_.Getpressao();
260     concentracao_ << dados_.Getconcentracao();
261
262     string texto_;
263
264     texto_ = atual_.Datastring() + horario_.Horastring() + " Temperatura: " +
265             temp_.str() + " Nivel: " + nivel_.str() + " Vazao: " + vazao_.str() +
266             " Pressao: " + pressao_.str() + " Concentracao: " + concentracao_.str() + "\n";
267
268     Hypnos_FILE.open("Logs_Sensores.txt", ios::app);
269     if(Hypnos_FILE.is_open()){
270         //cout << "Abertura do arquivo com Sucesso" << endl;
271         Hypnos_FILE << texto_;
272     }else{
273         cout << "Erro ao abrir o arquivo de texto" << endl;
274     }

```

Assim quando o programa é executado são gravadas, a priori, as restrições no topo do arquivo de saída e, após, cada linha tratada do arquivo de entrada e suas violações:

```

Logs_Sensores - Bloco de notas
Arquivo Editar Formatar Exibir Ajuda
-----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 18:01:25 Temperatura: 35.3 Nivel: 33.1 Vazao: 450.8 Pressao: 41.3 Concentracao: 70
-----ALERTA DE TEMPERATURA MAX-----^^^^^
-----ALERTA DE NIVEL MAX-----^^^^^
-----ALERTA DE VAZAO MAX-----^^^^^
-----ALERTA DE PRESSAO MAX-----^^^^^
-----ALERTA DE CONCENTRACAO MAX-----^^^^^
21/05/2018 18:01:25 Temperatura: 35.3 Nivel: 33.1 Vazao: 450.8 Pressao: 41.3 Concentracao: 70
-----ALERTA DE TEMPERATURA MAX-----^^^^^
-----ALERTA DE NIVEL MAX-----^^^^^
-----ALERTA DE VAZAO MAX-----^^^^^
-----ALERTA DE PRESSAO MAX-----^^^^^
-----ALERTA DE CONCENTRACAO MAX-----^^^^^

```

Arquivo de saída do teste gravação e Logs do tratamento de dados com violações

#### 4.1- Implementação classe Interface

```

3   #include "Openarquivo.h"
4
5
6   class Interface{
7       public:
8           Interface();
9           void Imprimeinterface();
10          ~Interface();
11
12      private:
13          Openarquivo abre_;
14  };
15

```

#### 4.2- Implementação main

O programa tem como finalidade o encapsulamento, proteção dos dados e do desenvolvimento da aplicação, ou seja, a segurança do projeto. Portanto, o arquivo main não fornece nenhuma informação sobre a implementação, torna somente visível a interface.

```

1   #include <iostream>
2   #include "Interface.h"
3
4   int main(){
5       Interface t;
6
7
8       return 0;
9   }
10

```

#### 5- Teste de unidade

Refere-se ao teste da menor parte testável de uma aplicação, pois devemos analisar as funções e seus procedimentos individualmente para assegurar sua integridade e se não há violações de memória.

Ao seguir os slides do curso tentamos realizar o teste de unidade a partir dos métodos das classes implementadas por meio do uso do framework Boost.test. Segue abaixo a implementação do teste de unidade realizado:

```

1 #define BOOST_TEST_MODULE Simple testcase
2 #include <C:/Program Files/boost/include/boost-1_67/boost/test/unit_test.hpp>
3
4 BOOST_AUTO_TEST_SUITE(suite01)
5
6 BOOST_AUTO_TEST_CASE(simple_test00)
7 {
8     BOOST_CHECK(Gethora() == 0);
9 }
10 BOOST_AUTO_TEST_CASE(simple_test01)
11 {
12     BOOST_CHECK(GetMinuto() == 0);
13 }
14 BOOST_AUTO_TEST_CASE(simple_test02)
15 {
16     BOOST_CHECK(GetSegundo() == 0);
17 }
18 BOOST_AUTO_TEST_CASE(simple_test03)
19 {
20     BOOST_CHECK(Horastring() == "00:00:00");
21 }
22
23 BOOST_AUTO_TEST_SUITE_END()
24

```

Ao compilar o código acima ocorre o seguinte erro.

```

C:\Users\leunao\Documents\UFMG\Eng_Sistemas\PDS_2\Tp_Final>g++ -c test.cpp
In file included from test.cpp:2:0:
C:/Program Files/boost/include/boost-1_67/boost/test/unit_test.hpp:18:37: fatal error: boost/test/test_tools.hpp: No such file or directory
 #include <boost/test/test_tools.hpp>
                                         ^
compilation terminated.

```

Em razão de não saber como corrigir tal erro, a equipe decidiu realizar os testes de forma diferente, como está definido nos itens abaixo:

### 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

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 // << "Total de Dias eh: " << B.total_de_dias() << endl;
21
22 //Testa para saber se a data da esquerda é a menor
23 int bb = 9; //colocar 9 apenas para testar se a var bb será 0 ou 1 da faze
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 //Testa 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
```

Teste de impressão na tela da sobrecarga de operador na tela:

```
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 o teste se refere a apenas um dia:

```
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(27);
15     B.SetMes(5);
16     B.SetAno(2016);
```

```

40     //Teste da 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 tem entre duas datas

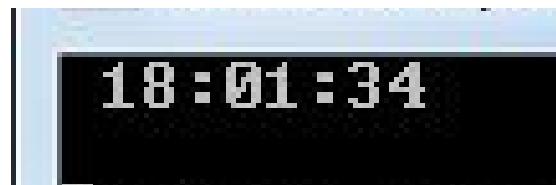
## 5.2- Teste da Classe Hora

Teste dos métodos Set e Get para alterações dos valores da classe Hora:

```

53
54     Hora teste_;
55
56     teste_.SetHora(18);
57     teste_.SetMinuto(1);
58     teste_.SetSegundo(34);
59
60     teste_.Imprime_Tempo();
61

```



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

```
3
4     int main(){
5         //Interface t;
6         Openarquivo teste_;
7         teste_.Abrearchivo();
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

```
3
4     int main(){
5         //Interface t;
6         Openarquivo teste_;
7         teste_.Abrearchivo();
8         teste_.Contagemlinhas();
9         //teste_.Mostraarquivo();
10        //abre_.Tratamento();
11        teste_.Fechaaquivo();
```

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!  
Quantidade de linhas eh: 3  
---->Fechamento do arquivo realizado com sucesso!  
Process returned 0 <0x0> execution time : 3.467 s  
Press any key to continue.
```

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 leitura da temperatura se o sensor
75 //não identificar alguma alteração da 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 }
```

```

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!

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!

Process returned 0 (0x0)   execution time : 3.778 s
Press any key to continue.
```

## 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_);

```

```

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!

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!

Process returned 0 (0x0)   execution time : 4.961 s
Press any key to continue.

```

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     //Separahorario(date_);
86     Separahorario(hour_);
87     //Separavalores(value_);
88     //AnalisaValores(); //Verifica se os valores correspondem entre os limites superiores e inferiores
89
90 }
91

```

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

```

```

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!

Imprimido horario: 18:01:25
HORA CONUERTIDA:
Hora: 18 Minuto: 1 Segundos: 25

Imprimido horario: 18:02:34
HORA CONUERTIDA:
Hora: 18 Minuto: 2 Segundos: 34

Imprimido horario: 18:04:45
HORA CONUERTIDA:
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
85     //Tratamento das informações
86     //Separahorario(date_);
87     //Separahorario(hour_);
88     Separavalores(value_);
89
90 }

```

///Analisavalores(); //Verifica se os valores correspondem entre os limites superiores e inferiores

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;
169

```

```

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!

Imprimindo valores: 33.1;450.8;35.3;41.3;70;
VALORES CONVETIDOS:
Nivel: 33.1 Vazao: 450.8 Temp: 35.3 Press: 41.3Conc: 70
Imprimindo valores: 32.8;432.3;25.3;46.3;90;
VALORES CONVETIDOS:
Nivel: 32.8 Vazao: 432.3 Temp: 25.3 Press: 46.3Conc: 90
Imprimindo valores: 35.1;442.9;23.3;43.3;40;
VALORES CONUERTIDOS:
Nivel: 35.1 Vazao: 442.9 Temp: 23.3 Press: 43.3Conc: 40
----->Fechamento do arquivo realizado com sucesso!

Process returned 0 (0x0)   execution time : 5.630 s
Press any key to continue.

```

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:

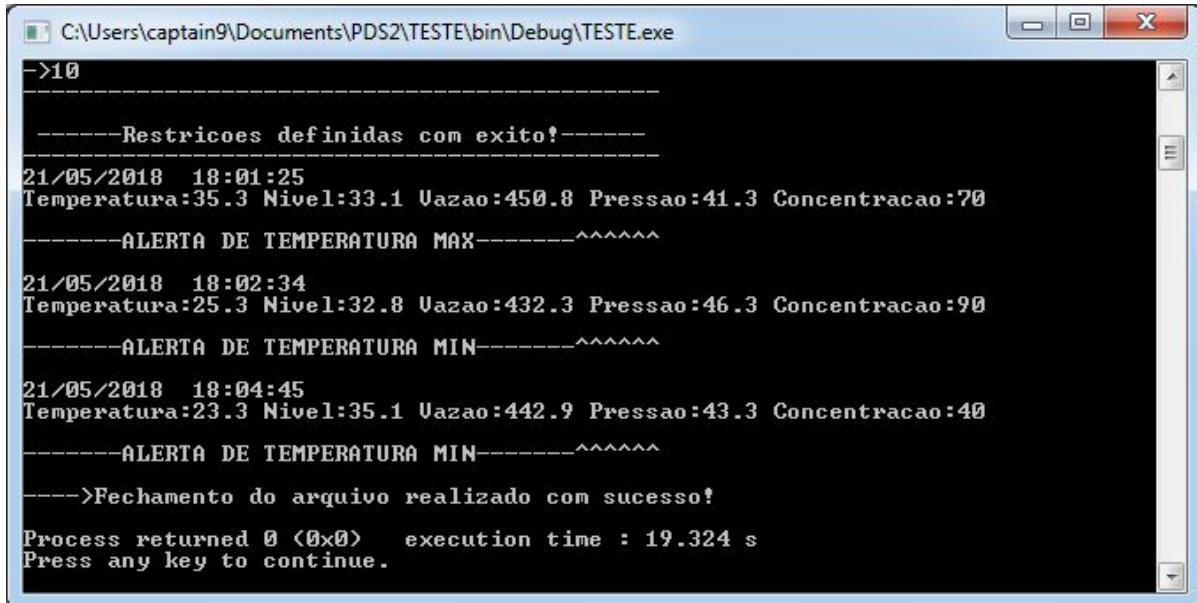
```

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     }

```

A utilização de if's em vez do Try-Catch se deve a possibilidade de haver mais de uma violação de restrição, pois quando o programa identifica a exceção, quando utilizamos o tratamento de exceção, ele para a execução da linha de código e realiza o tratamento, porém, o programa não verifica se existe outras violações do sistema. A informação perdida

no tratamento é importante para o usuário, pois saber quais itens da fábrica que precisam de manutenção.

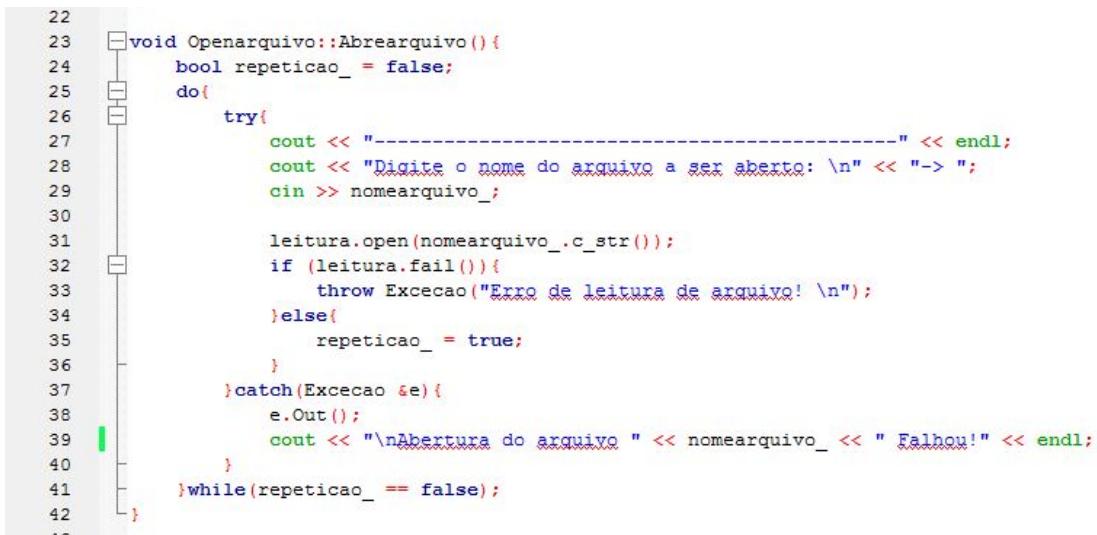


```
C:\Users\captain9\Documents\PDS2\TESTE\bin\Debug\TESTE.exe
>10
-----Restricoes definidas com exito!-----
21/05/2018 18:01:25
Temperatura:35.3 Nivel:33.1 Uazao:450.8 Pressao:41.3 Concentracao:70
-----ALERTA DE TEMPERATURA MAX-----^^^^^^^
21/05/2018 18:02:34
Temperatura:25.3 Nivel:32.8 Uazao:432.3 Pressao:46.3 Concentracao:90
-----ALERTA DE TEMPERATURA MIN-----^^^^^^^
21/05/2018 18:04:45
Temperatura:23.3 Nivel:35.1 Uazao:442.9 Pressao:43.3 Concentracao:40
-----ALERTA DE TEMPERATURA MIN-----^^^^^^^
---->Fechamento do arquivo realizado com sucesso!
Process returned 0 <0x0> execution time : 19.324 s
Press any key to continue.
```

Teste de função para restrição de Temperatura máxima e mínima

## 6-Tratamento de Exceções

O tratamento de exceção foi implementado em duas funções: na função de abertura de arquivo (AbreArquivo()) e no construtor da classe Exceção:



```
22
23     void Openarquivo::AbreArquivo(){
24         bool repeticao_ = false;
25         do{
26             try{
27                 cout << "-----" << endl;
28                 cout << "Digite o nome do arquivo a ser aberto: \n" << "-> ";
29                 cin >> nomearquivo_;
30
31                 leitura.open(nomearquivo_.c_str());
32                 if (leitura.fail()){
33                     throw Excecao("Erro de leitura de arquivo! \n");
34                 }else{
35                     repeticao_ = true;
36                 }
37             }catch(Excecao &e){
38                 e.Out();
39                 cout << "\nAbertura do arquivo " << nomearquivo_ << " Falhou!" << endl;
40             }
41         }while(repeticao_ == false);
42     }
```

Caso o usuário digite o nome errado do arquivo de entrada o programa não encerra e volta a pedir o nome do arquivo correto:

```

C:\Users\captain9\Documents\PDS2\TESTE\bin\Debug\TESTE.exe

Digite o nome do arquivo a ser aberto:
> Sensor1.ttx
Erro de leitura de arquivo!

Abertura do arquivo Sensor1.ttx Falhou!
Digite o nome do arquivo a ser aberto:
> Sensor1.txt
----- Restricoes do Sistema -----
----- Restricoes de Temperatura Maxima:-----
>

```

Exibe a notificação do tratamento da exceção que a abertura do arquivo falhou e reinicia o processo.

No construtor da classe Exceção há a inicialização dos atributos das restrições de limite superior e inferior, nesta parte é necessária a implementação do tratamento de exceção, pois para o funcionamento do programa é necessário que os valores fornecidos pelo usuário sejam maiores que zero, no caso dos limites superiores, e maiores que zero e menores que o limite superior, no caso do limite inferior. Dessa forma, foi implementado o algoritmo parecido ao tratamento de exceção da função AbreArquivo():

```

13     vazaoinferior_ = 0.;
14     pressaosuperior_ = 0.;
15     concentracaosuperior_ = 0.;
16     concentracaoinferior_ = 0.;
17     cout << "-----" << endl;
18     cout << "-----Restricoes do Sistema-----" << endl;
19     bool repeticao_ = false;
20     do{
21         try{
22             cout << "-----Restricoes de Temperatura Maxima:-----\n" << "->";
23             cin >> tempsuperior_;
24             if(tempsuperior_ <= .0){
25                 throw Excecao("-----TEMPETURA SUPERIOR invalido!-----\n");
26             }else{
27                 repeticao_ = true;
28             }
29         }catch(Excecao &e){
30             e.Out();
31             cout << "-----Entre com um valor maior que ZERO para Temperatura Maxima-----" << endl;
32         }
33     }while(repeticao_ == false);

//volta a variavel repeticao para o valor inicial
repeticao_ = false;

```

A função possui uma variável booleana que realiza a saída do programa da função Do-while, pois enquanto a variável repeticao\_ for falsa essa parte do programa será repetida até que o usuário coloque um valor válido. O limite superior é definido com um número do tipo double acima de .0 (O sistema está restrito com números positivos) e em seguida é solicitado o número referente ao limite inferior, lembrando que a variável restricao\_ volta para o status de false, para que a função possa ter o looping correto do próximo Do-While. O limite inferior deve ser maior que zero e menor que a variável superior, limitando assim seu valor ao sistema.

```

C:\Users\captain9\Documents\PDS2\TESTE\bin\Debug\TESTE.exe

|||||BEM VINDO AO SENSOR|||||
-----
Digite o nome do arquivo a ser aberto:
-> Sensor1.txt
----- Restricoes do Sistema -----
----- Restricoes de Temperatura Maxima:-----
->40 ----- Restricoes de Temperatura Minima:-----
->0 ----- TEMPERATURA INFERIOR invalido!-----
----- Entre com um valor diferente de ZERO e menor que a TEMPERATURA SUPERIOR-----
----- Restricoes de Temperatura Minima:-----
->55 ----- TEMPERATURA INFERIOR invalido!-----
----- Entre com um valor diferente de ZERO e menor que a TEMPERATURA SUPERIOR-----
----- Restricoes de Temperatura Minima:-----
->

```

Teste da função com as restrições

## 7- Git e GitHub

É essencial a utilização do Git e GitHub para controle de versão do programa, pois anteriormente era feito backup manual dos arquivos via nuvem, após a aula do monitor Marcos, podemos implementar a utilização de repositórios online e retorno dos commits pelo site GitHub.

Endereço onde os commits estão salvos na branch master:

[https://github.com/igorpetrucci/TP\\_PDS2\\_20182](https://github.com/igorpetrucci/TP_PDS2_20182)

## 8- Implementação para arquivo de saída e impressão na tela

Interface faz o agrupamento das funções principais e promove o encapsulamento dos dados:

```

2
3     Interface::Interface() {
4         Imprimeinterface();
5         abre_.Abreearquivo();
6         abre_.Contagemlinhas();
7         abre_.Definerestricoes();
8         abre_.Tratamento();
9         abre_.Fechaaquivo();
10    }
11
12
13 void Interface::Imprimeinterface() {
14     cout << "-----" << endl;
15     cout << "|||||||BEM VINDO AO SENSOR|||||||" << endl;
16     cout << "-----" << endl;
17     cout << "-----" << endl;
18     cout << "-----" << endl;
19
20

```

O construtor da interface realiza a chamada de cada etapa da leitura de dados e tratamento de informações.

C:\Users\captain9\Documents\PDS2\TESTE\bin\Debug\TESTE.exe

```
BEM VINDO AO SENSOR

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

-----Restricoes do Sistema-----
-----Restricoes de Temperatura Maxima-----
->40 -----Restricoes de Temperatura Minima-----
->30 -----Restricoes de Nivel Maximo-----
->40 -----Restricoes de Nivel Minimo-----
```

Exemplo de interface

## 9- Visão geral do arquivo de entrada, interface e arquivo de saída

Realizado o teste com vários logs de entrada a fim de verificar a eficiência do programa e estabilidade:

Sensor1 - Bloco de notas

	Arquivo	Editar	Formatar	Exibir	Ajuda
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;					
2018/05/21 18:10:25 33.1;450.8;35.3;41.3;70;					
2018/05/21 18:11:34 32.8;432.3;25.3;46.3;90;					
2018/05/21 18:32:45 35.1;442.9;23.3;43.3;40;					
2018/05/21 18:01:25 33.1;450.8;35.3;41.3;70;					
2018/05/21 19:02:34 32.8;432.3;25.3;46.3;90;					
2018/05/21 19:04:45 35.1;442.9;23.3;43.3;40;					
2018/05/22 08:01:25 33.1;450.8;35.3;41.3;70;					
2018/05/22 08:22:34 32.8;432.3;25.3;46.3;90;					
2018/05/22 08:54:45 35.1;442.9;23.3;43.3;40;					
2018/05/22 08:01:25 33.1;450.8;35.3;41.3;70;					
2018/05/23 08:22:34 32.8;432.3;25.3;46.3;90;					

Arquivo de entrada

```
C:\Users\captain9\Documents\PDS2\TESTE\bin\Debug\TESTE.exe
|||||BEM VINDO AO SENSOR|||||
-----
Digite o nome do arquivo a ser aberto:
-> Sensor1.txt
-----Restricoes do Sistema-----
-----Restricoes de Temperatura Maxima-----
->40
-----Restricoes de Temperatura Minima-----
->30
-----Restricoes de Nivel Maximo-----
->40
-----Restricoes de Nivel Minimo-----
->30
-----Restricoes de Vazao Maxima-----
->500
-----Restricoes de Vazao Minima-----
->100
-----Restricoes de Pressao Maxima-----
->70
-----Restricoes de Pressao Minima-----
->30
-----Restricoes de Concentracao Maxima-----
->100
-----Restricoes de Concentracao Minima-----
->50
-----Restricoes definidas com exito!
-----
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-----
```

Visão geral do programa:

```
Logs_Sensores - Bloco de notas
Arquivo Editar Formatar Exibir Ajuda
-----RESTRICOES-----
Temperatura superior: 40
Temperatura inferior: 10
Nivel superior: 50
Nivel inferior: 40
Vazao superior: 40
Vazao inferior: 30
Pressao superior: 10
Pressao inferior: 5
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
-ALERTA DE NIVEL MIN-^^^^^
-ALERTA DE VAZAO MAX-----^^^^^
-----ALERTA DE PRESSAO MAX-----^^^^^
21/05/2018 18:02:34 Temperatura: 25.3 Nivel: 32.8 Vazao: 432.3 Pressao: 46.3 Concentracao: 90
-ALERTA DE NIVEL MIN-^^^^^
-ALERTA DE VAZAO MAX-----^^^^^
-----ALERTA DE PRESSAO MAX-----^^^^^
21/05/2018 18:04:45 Temperatura: 23.3 Nivel: 35.1 Vazao: 442.9 Pressao: 43.3 Concentracao: 40
-ALERTA DE NIVEL MIN-^^^^^
-ALERTA DE VAZAO MAX-----^^^^^
-----ALERTA DE PRESSAO MAX-----^^^^^
-----ALERTA DE CONCENTRACAO MIN-----^^^^^
21/05/2018 Temperatura: 35.3 Nivel: 33.1 Vazao: 450.8 Pressao: 41.3 Concentracao: 70
-ALERTA DE NIVEL MIN-^^^^^
-ALERTA DE VAZAO MAX-----^^^^^
-----ALERTA DE PRESSAO MAX-----^^^^^
21/05/2018 18:11:34 Temperatura: 25.3 Nivel: 32.8 Vazao: 432.3 Pressao: 46.3 Concentracao: 90
-ALERTA DE NIVEL MIN-^^^^^
-ALERTA DE VAZAO MAX-----^^^^^
-----ALERTA DE PRESSAO MAX-----^^^^^
-----ALERTA DE CONCENTRACAO MIN-----^^^^^
21/05/2018 18:01:25 Temperatura: 35.3 Nivel: 33.1 Vazao: 450.8 Pressao: 41.3 Concentracao: 70
-ALERTA DE NIVEL MIN-^^^^^
-ALERTA DE VAZAO MAX-----^^^^^
-----ALERTA DE PRESSAO MAX-----^^^^^
21/05/2018 18:32:45 Temperatura: 23.3 Nivel: 35.1 Vazao: 442.9 Pressao: 43.3 Concentracao: 40
-ALERTA DE NIVEL MIN-^^^^^
-ALERTA DE VAZAO MAX-----^^^^^
-----ALERTA DE PRESSAO MAX-----^^^^^
-----ALERTA DE CONCENTRACAO MIN-----^^^^^
21/05/2018 19:02:34 Temperatura: 25.3 Nivel: 32.8 Vazao: 432.3 Pressao: 46.3 Concentracao: 90
-ALERTA DE NIVEL MIN-^^^^^
-ALERTA DE VAZAO MAX-----^^^^^
-----ALERTA DE PRESSAO MAX-----^^^^^
21/05/2018 19:04:45 Temperatura: 23.3 Nivel: 35.1 Vazao: 442.9 Pressao: 43.3 Concentracao: 40
-ALERTA DE NIVEL MIN-^^^^^
-ALERTA DE VAZAO MAX-----^^^^^
-----ALERTA DE PRESSAO MAX-----^^^^^
-----ALERTA DE CONCENTRACAO MIN-----^^^^^
22/05/2018 8:01:25 Temperatura: 35.3 Nivel: 33.1 Vazao: 450.8 Pressao: 41.3 Concentracao: 70
```

#### Arquivo de saída

O programa seguiu o procedimento conforme esperado.

### 10-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.



