Métodos de Programação:
Trabalho 3 - Contador de Código em C++

Arthur da Veiga Feitoza Borges, 13/0050725

October 11, 2018

1 Descrição do Trabalho

1.1 Arquivos referentes ao Contador de Código

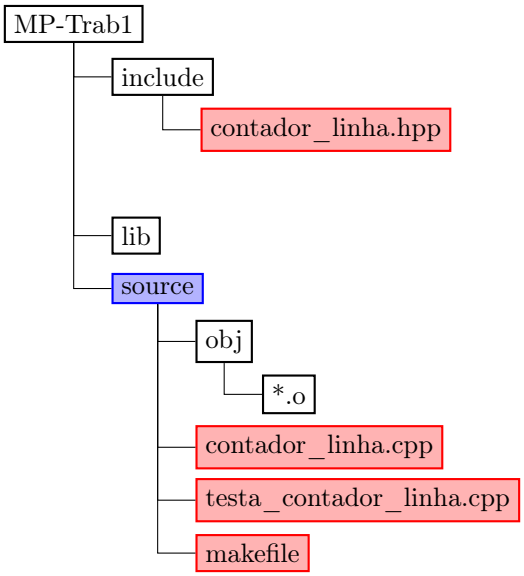

Figure 1: Organização dos diretórios do Trabalho 1

Dada a organização dos diretórios e arquivos do trabalho, falaremos mais detalhadamente dos arquivos que estão demarcados em vermelho. O diretório demarcado em azul é onde estaremos sempre, para poder manejar este trabalho apropriadamente (falaremos disso ao longo deste documento).

1.1.1 ../include/contador_linha.hpp

O arquivo contador_linha.hpp consiste na declaração da enum STATE, que mostra os estados da máquina de estado que foi implementada em contador_linha.cpp e as funções para o funcionamento da pilha. Nele temos o seguinte:

- Definição para header:

```
1 #ifndef INCLUDE_CONTADOR_LINHA_HPP_
2 #define INCLUDE_CONTADOR_LINHA_HPP_
```

- As bibliotecas que foram usadas:

```
1 #include <iostream>
2 #include <fstream>
3 #include <string>
4 #include <sstream>
5 #include <utility>
```

- A estrutura utilizada para representar a máquina de estados:

```
1  enum STATE {
2      init,
3      espaco,
4      barra,
5      barra_dupla,
6      barra_invertida,
7      barra_asterisco,
8      barra_asterisco_asterisco,
9      barra_asterisco_asterisco_barra
10 };
```

  Onde:

  - **init:** é o estado inicial da máquina.
  - **espaco:** é o estado no qual temos uma linha só com espaços
  - **barra:** o estado no qual temos uma barra. É o estado de entrada para os comentários.
  - **barra_dupla:** o estado no qual temos duas barras, que é comentário de única linha.
  - **barra_invertida:** o estado no qual temos duas barras e uma outra invertida, que são comentários de única linha adicionado de uma linha extra.
  - **barra_asterisco:** aqui já temos o estado no qual temos uma barra e um asterisco, que é o comentário de múltiplas linhas.
  - **barra_asterisco_asterisco:** aqui já temos o estado no qual temos uma barra e dois asteriscos, que é comentário de múltiplas linhas.
  - **barra_asterisco_asterisco_barra:** é o último estado, que é o comentário de múltiplas linhas já fechado.

- As funções declaradas. Detalharemos o que elas fazem em contador_linha.cpp:

```
1  int abre_arquivo(std::filebuf *file, std::string file_name);
2  std::pair <int, int> count_linhas(std::string fileString);
3  int fecha_arquivo(std::filebuf *file);
4  std::pair <int, int> le_arquivo_to_string_count(std::string file_name);
```

1.1.2 contador_linha.cpp

Neste arquivo temos as implementações do que foi declarado em contador_linha.hpp. Tudo o que foi declarado em contador_linha.hpp é chamado com #include. O contador_linha.cpp consiste de:

- **abre_arquivo**

```
1  int abre_arquivo(std::filebuf * file, std::string file_name) {
2      if (file_name == "") {
3          return -1;
4      } else {
5          if (!file->is_open()) {
6              file->std::filebuf::open(file_name, std::fstream::in);
7              if (file->is_open()) {
8                  return 0;
9              }
10         }
11         return -1;
12     }
13 }
```

  - **Detalhamento da Função:** A função abre_arquivo é simples. Ele abre um arquivo válido no buffer para leitura/escrita. No caso usaremos somente para leitura. Nele checamos se o arquivo existe e se ele já estava aberto. Se um dos dois casos acontece, é retornado -1, indicando que houve falha.
  - **Assertivas de Entrada:** std::filebuf * file, std::string file_name.
  - **Assertivas de Saída:** std::filebuf * file, o valor retornado pela função.
  - **Parâmetros de Entrada:**
    - std::filebuf * file: o ponteiro do buffer no qual o arquivo será aberto.
    - std::string file_name: o nome do arquivo que será aberto.
  - **Parâmetros de Saída:** um int que retorna se o procedimento teve sucesso ou não.
  - **Exceções:**
    - Se o file_name estiver vazio;
    - Se o arquivo já estiver aberto no buffer.

- **count_linhas**

```
1  std::pair <int, int> count_linhas(std::string fileString) {
2      if (fileString == "") {
3          return std::make_pair(1, 0);
4      }
5      int count = 0, line_number = 0;
6      STATE estado = init;
7      std::stringstream data_stream(fileString);
8      std::string line;
9      while (std::getline(data_stream, line)) {
10         std::stringstream line_stream(line);
11         char line_char;
12         while (line_stream) {
13             line_char = line_stream.get();
14             switch (estado) {
15                 case init:
16                     if (line.size() == 0)
17                         estado = espaco;
18                     if (line_char != ' ') {
19                         if (line_char != '/') {
20                             while (line_stream) {
21                                 line_char = line_stream.get();
22                             }
23                         } else {
24                             estado = barra;
25                         }
26                     } else {
27                         line.erase(line.find_last_not_of(" \n\r\t")+1);
28                         if (line.size() == 0)
29                             estado = espaco;
30                     }
31                     break;
32                 case espaco:
33                     break;
34                 case barra:
35                     if (line_char == '/') {
36                         estado = barra_dupla;
37                     } else {
38                         if (line_char == '*') {
39                             estado = barra_asterisco;
40                         } else {
41                             estado = init;
42                         }
43                     }
44                     break;
45                 case barra_dupla:
46                     if (line.back() == '\\') {
47                         estado = barra_invertida;
48                     }
49                     break;
50                 case barra_invertida:
51                     if (line.back() != '\\') {
52                         estado = barra_dupla;
53                     }
54                     break;
55                 case barra_asterisco:
56                     if (line_stream) {
57                         if (line_char == '*') {
```

```
58                              estado = barra_asterisco_asterisco;
59                              break;
60                          }
61                      }
62                      break;
63                  case barra_asterisco_asterisco:
64                      if (line_stream) {
65                          if (line_char == '/') {
66                              estado = barra_asterisco_asterisco_barra;
67                              break;
68                          }
69                      }
70                      break;
71                  case barra_asterisco_asterisco_barra:
72                      if (line_stream)
73                          estado = init;
74                      line.erase(line.find_last_not_of(" \n\r\t")+1);
75                      if (line.size() == 0)
76                          estado = espaco;
77                      break;
78              }
79          }
80          line_number++;
81          switch (estado) {
82              /* casos de init e barra, conta mais uma linha e define estado = init;*/
83              case init:
84              case barra:
85                  count++;
86                  estado = init;
87                  break;
88              /* casos de espaco e barra_dupla, define estado = init somente;*/
89              case espaco:
90              case barra_dupla:
91                  estado = init;
92                  break;
93          }
94      }
95      return std::make_pair(0, count);
96  }
```

- **Detalhamento da Função:** A função count_linhas é o coração deste projeto. É nele que contamos quantas linhas um programa em C++ tem. Esta função consiste em três etapas:

  1. Organização de fileString para duas stringstreams cascateadas, levando ao nível de char (arquivo -> linha -> char), onde a primeira tem o arquivo completo e a segunda tem uma linha de cada vez. Essa distribuição é feita da seguinte forma: fileString (string - arq) -> data_stream (sstream - arq) -> line (string - linha) -> line_stream (sstream - linha) -> line_char (char);
  2. Manipulação de line_char para fazer o tratamento das linhas de código. Esta e a próxima etapa andam juntas: elas formam uma máquina de estados eficiente para fazer as devidas classificações das linhas para ver a linha é contada ou não. E;

3. Contagem, tendo como base o estado resultante. A contagem só acontece nos estados init e barra.

- **Assertivas de Entrada:** std::string fileString, a enumeração STATE definida no .hpp,
- **Assertivas de Saída:** int count, no par retornado pela função.
- **Parâmetros de Entrada:** std::string fileString: o arquivo completo em forma de string.
- **Parâmetros de Saída:** std::pair <int, int> que retorna um par onde:
  - O primeiro elemento serve para ver se o procedimento teve sucesso e;
  - O segundo elemento é o número de linhas resultantes da contagem de count_linhas.
- **Exceções:** Não há exceções que causem erro.

É importante ressaltar que nesta função existe uma máquina de estado para o tratamento das linhas. Na próxima seção a detalharemos.

- **fecha_arquivo**

```
1  int fecha_arquivo(std::filebuf *file) {
2      if (file == NULL)
3          return -1;
4      if (file->is_open()) {
5          file->close();
6          return 0;
7      }
8      return -1;
9  }
```

- **Detalhamento da Função:** A função fecha_arquivo é simples. Ele fecha o buffer no qual está aberto o arquivo para leitura/escrita.
- **Assertivas de Entrada:** std::filebuf * file
- **Assertivas de Saída:** std::filebuf * file, o valor retornado pela função.
- **Parâmetros de Entrada:** std::filebuf * file: é o ponteiro do buffer de arquivo que vamos fechar.
- **Parâmetros de Saída:** int que retorna se o procedimento teve sucesso ou não.
- **Exceções:**
  - Se o buffer é nulo;
  - Se o arquivo já está fechado.

- **le_arquivo_to_string_count**

```
1  std::pair <int, int> le_arquivo_to_string_count(std::string file_name) {
2      int count = 0, check = -1;
3      if (file_name == "")
4          return std::make_pair(-1, count);
5      std::filebuf file;
6      check = abre_arquivo(&file, file_name);
7      if (check == 0) {
8          std::stringstream fileInput("");
9          fileInput << &file;
10         std::string fileString(fileInput.str());
11         count = count_linhas(fileString).second;
12         check = fecha_arquivo(&file);
13         if (check == 0) {
14             return std::make_pair(0, count);
15         }
16         return std::make_pair(-1, count);
17     }
18     return std::make_pair(-1, count);
19 }
```

- **Detalhamento da Função:** A função le_arquivo_string_count é a função que o cliente usará. Ele faz todo o procedimento de abertura do arquivo, contagem das linhas e fechamento do mesmo.
- **Assertivas de Entrada:** std::string file_name, std::filebuf file.
- **Assertivas de Saída:** somente o std::make_pair(check, count).
- **Parâmetros de Entrada:** file_name é o nome do arquivo que se quer que conte as linhas, desconsiderando comentários e espaços.
- **Parâmetros de Saída:** std::pair <int, int> que retorna um par onde:
  - O primeiro elemento serve para ver se o procedimento teve sucesso e;
  - O segundo elemento é o número de linhas resultantes da contagem de count_linhas.
- **Exceções:**

1.1.3 count_linhas: Máquina de estados

Falaremos brevemente de cada estado:

- **case init:** é o estado inicial da máquina. observam-se as seguintes situações:
  - line_char é '/' -> estado = barra;
  - line_char é ' ' -> deleta-se de line as barras de espaço e checa se o seu tamanho é zero. Se sim, estado = espaço;
  - o tamanho da line é zero -> estado = espaco;
- **case espaco:** aqui não se faz nada, só na hora da contagem da linha.

- **case barra:** aqui já temos o estado no qual temos uma barra . observam-se as seguintes situações:
  - line_char é '/' -> estado = barra_dupla, que são comentários de única linha.
  - line_char é '*' -> estado = barra_asterisco, que são comentários de múltiplas linhas.
  - se não for nenhum dos casos -> estado = init.
- **case barra_dupla:** aqui já temos o estado no qual temos duas barras, que é comentário de única linha. observa-se a seguinte situação:
  - se o último caractere de line é barra invertida, estado = barra_invertida, que são comentários de única linha adicionado de uma linha extra.
  - se não acontece a situação acima, o estado é mantido.
- **case barra_invertida:** aqui já temos o estado no qual temos duas barras e uma outra invertida, que são comentários de única linha adicionado de uma linha extra. observa-se as seguintes situações:
  - se o último caractere de line não é barra invertida -> estado = barra_dupla, que são comentários de única linha.
  - se não acontece a situação acima, o estado é mantido.
- **case barra_asterisco:** aqui já temos o estado no qual temos uma barra e um asterisco, que é comentário de múltiplas linhas. observa-se as seguintes situações.
  - se ainda há elementos em line_stream E line_char é '*' -> estado = barra_asterisco_asterisco, que ainda são comentários de múltiplas linhas.
  - se não acontece a situação acima, o estado é mantido.
- **case barra_asterisco_asterisco:** aqui já temos o estado no qual temos uma barra dois asteriscos, que é comentário de múltiplas linhas. observa-se as seguintes situações.
  - se ainda há elementos em line_stream E line_char é '/' -> estado = barra_asterisco_asterisco_barra, que é quando o comentário de múltiplas linhas é fechado.
  - se não acontece a situação acima, o estado é mantido.
- **case barra_asterisco_asterisco_barra:** aqui fechamos o comentário de múltiplas linhas. Mas é importante observar que:
  - se ainda há elementos em line_stream, estado = init. Isso faz com que a máquina de estados resete.
  - mas, se deleta-se de line as barras de espaço e, se o seu tamanho é zero é verdadeiro, estado = espaco

1.1.4 testa_contador_linha.cpp

Este arquivo tem todos os testes diretamente relacionados às funções implementadas no pilha.cpp. Estes testes foram implementados antes de criar as funções no .cpp em si.

- **TEST_CASE: abre_arquivo** - Caso de teste para a função abre_arquivo. Aqui foram feitos os seguintes testes:
  - **SECTION("abre_arquivo: ERRO_stringVazia"):** teste quando tenta abrir arquivo sem um nome.
  - **SECTION("abre_arquivo: ERRO_arquivoNaoExiste"):** teste quando se abre um arquivo que não existe no diretório.
  - **SECTION("abre_arquivo: ERRO_arquivoJaAberto"):** teste quando se tenta abrir um arquivo já aberto.
  - **SECTION("abre_arquivo: OK"):** teste do funcionamento normal da função.
- **TEST_CASE: count_linhas** - Caso de teste para a função count_linhas. Aqui foram feitos os seguintes testes:
  - **SECTION("count_linhas: OK_arquivoVazio"):** teste quando se conta um arquivo vazio.
  - **SECTION("count_linhas: OK_semComentarios"):** teste quando se conta um arquivo sem comentário algum. Esse teste é focado para os estados init e espaco.
  - **SECTION("count_linhas: OK_comComentariosBarraDupla"):** teste quando se conta um arquivo com comentários de barra dupla. Esse teste é focado para os estados barra, barra_dupla e barra_invertida.
  - **SECTION("count_linhas: OK_comComentariosBarraAsterisco"):** teste quando se conta um arquivo com comentários de barra dupla. Esse teste é focado para os estados barra, barra_asterisco, barra_asterisco_asterisco e barra_asterisco_asterisco_barra.
  - **SECTION("count_linhas: OK_comComentariosTotal"):** teste quando se conta um arquivo com comentários quaisquer. Esse teste é focado para todos os estados.
- **TEST_CASE: fecha_arquivo** - Caso de teste para a função fecha_arquivo. Aqui foram feitos os seguintes testes:
  - **SECTION("fecha_arquivo: ERRO_naoHaFilebuf"):** teste quando tenta-se fechar um buffer de arquivo que não existe.
  - **SECTION("fecha_arquivo: ERRO_arqNaoExiste"):** teste quando tenta-se fechar um arquivo que não existe, por tanto, um arquivo que não foi aberto.

  - **SECTION("fecha_arquivo: OK"):** teste do funcionamento normal da função.

- **TEST_CASE: le_arquivo_to_string_count** - Caso de teste para a função le_arquivo_to_string_count. Aqui foram feitos os seguintes testes:
  - **SECTION("le_arquivo_to_string_count: ERRO_stringVazia"):** teste quando tenta abrir arquivo sem um nome.
  - **SECTION("le_arquivo_to_string_count: ERRO_arqNaoExiste"):** teste quando se abre um arquivo que não existe no diretório.
  - **SECTION("le_arquivo_to_string_count: OK"):** teste do funcionamento normal da função.

2 Passos/Procedimentos para execução dos testes

Antes de iniciarmos, as seguintes instalações devem ser feitas:

- Make: `sudo apt-get install make`
- Catch: `sudo apt-get install catch`
- C++: `sudo apt-get update && sudo apt-get install build-essential`

Para compilar e executar o módulo Pilha, os seguintes comandos são necessários **com o terminal aberto dentro da pasta /source com os códigos**, sequencialmente:

1. `clear && make clean && make`
2. `make run`