Documentação Contador de Linhas

1.0

Gerado por Doxygen 1.8.14

Sumário

Capítulo 1

Índice dos Arquivos

1.1 Lista de Arquivos

Esta é a lista de todos os arquivos e suas respectivas descrições:

Capítulo 2

Arquivos

2.1 Referência do Arquivo conta_linhas/testes/amostra1.cpp

Explicação

Arquivo criado para servir de teste para a função principal de contagem de linhas. É um programa simples que avalia se um triângulo cujos lados foram fornecidos pelo usuário é retângulo ou não.

2.1.1 Funções

2.1.1.1 main()

```
void main ( )
```

```
16              {
17     double cateto1, cateto2, hipot;    // comentario in-line
18
19     printf("Insira os valores dos catetos e da hipotenusa.\n");
20     scanf("%lf %lf %lf", &cateto1, cateto2, hipot);
21
22     if ((cateto1 * cateto1) + (cateto2 * cateto2) == (hipot * hipot)) {
23         printf("Triangulo retangulo valido.\n");
24     } else {
25         printf("Triangulo retangulo invalido.\n");
26     }
27
28     // comentario entre linhas de codigo
29
30     return 0;
31 }
```

2.2 Referência do Arquivo conta_linhas/testes/amostra2.cpp

Explicação

Arquivo criado para servir de teste para a função principal de contagem de linhas. É um programa simples que mostra no terminal o número que o usuário forneceu com os algarismos em ordem inversa.

2.2.1 Funções

2.2.1.1 main()

```
int main ( )
```

```
9           {
10      int n;
11      scanf("%d", &n);
12
13      if (n < 0) {
14          n = -n;
15          printf("-");
16      }
17
18      while (n > 0) {
19          printf("%d", n%10);
20          n /= 10;
21      }
22
23      printf("\n");
24
25      return 0;
26  }
```

2.3 Referência do Arquivo conta_linhas/testes/amostra3.cpp

Funções

Arquivo criado para servir de teste para a função principal de contagem de linhas. É um programa simples que conta o número de caracteres numa string fornecida pelo usuário.

2.3.1 Funções

2.3.1.1 main()

```
int main ( )
```

```
6           {
7       char string[80];
8       int i = 0;
9       scanf("%s", string);
10       while (string[i] != '\0') {
11          i++;
12      }
13      printf("%d\n", i);
14      return 0;
15  }
```

2.4 Referência do Arquivo conta_linhas/testes/conta_linhas.c

```
#include <stdio.h>
#include <ctype.h>
#include <stdbool.h>
#include "conta_linhas.h"
```

Funções

- int conta_linhas_codigo (FILE *codigo)
- int checa_vazia (char *linha, const int buffer)
- int checa_comentario (char *linha, const int buffer, bool *flag_comentario)
- int tamanho_linha (char *linha, const int buffer)

2.4.1 Funções

2.4.1.1 checa_comentario()

```
int checa_comentario (
            char * linha,
            const int buffer,
            bool * flag_comentario )
```

Faz uso da flag_comentario para distinguir entre comentário de linha única e bloco de comentário.

Caso não esteja em bloco de comentário, pula todo caracter "whitespace" e analisa os dois primeiros válidos, para determinar se há um início de bloco de comentário nessa linha.

Caso esteja em bloco de comentário, analisa todos os caracteres da linha em pares, verificando se há ocorrência de término de bloco de comentário.

Retornos:
0, se linha não for de comentário.
1, se linha for de algum tipo de comentário.

Valores da flag_comentario:
0 -> não está em bloco de comentário atualmente.
1 -> está em bloco de comentário.

```
78                                                                          {
79     int tamanho = tamanho_linha(linha, buffer);
80     int contador = 0;
81
82     switch (*flag_comentario) {
83         case 0:
84             while (isspace(linha[contador]) && contador < tamanho) {
85                 contador++;
86             }
87
88             if (contador == tamanho || contador+1 == tamanho) {
89                 return 0;
90             } else if (linha[contador] == '/') {
91                 switch (linha[contador+1]) {
92                     case '*':
```

```
93                          *flag_comentario = 1;
94                          return 1;
95                      case '/':
96                          return 1;
97                      default:
98                          return 0;
99                  }
100             } else {
101                 return 0;
102             }
103
104         case 1:
105             for (contador = 0; contador+1 < tamanho; contador++) {
106                 if (linha[contador] == '*' && linha[contador+1] == '/') {
107                     *flag_comentario = 0;
108                 }
109             }
110             return 1;
111     }
112 }
```

2.4.1.2 checa_vazia()

```
int checa_vazia (
            char * linha,
            const int buffer )
```

Percorre o array que contém a linha a ser analizada e checa se há algum caracter que não seja classificado como "whitespace".

Retornos:
0, se linha contiver caracter não-"whitespace";
1, se linha for vazia.

```
44                                                  {
45     int tamanho = tamanho_linha(linha, buffer);
46
47     for (int i = 0; i < tamanho; i++) {
48         if (!isspace(linha[i])) {
49             return 0;
50         }
51     }
52
53     return 1;
54 }
```

2.4.1.3 conta_linhas_codigo()

```
int conta_linhas_codigo (
            FILE * codigo )
```

Lê o arquivo de código fornecido linha por linha e, para cada linha, checa se é uma linha vazia ou se é uma linha de comentário.

Faz essas ações por meio de outras funções: checa_vazia e checa_comentario.

Retorna o número de linhas válidas de código.

```
17                                      {
18     const int buffer = 100;
19     int linhas_codigo = 0;
20     bool flag_comentario = 0;
21     char linha[buffer];
22
23     while (fgets(linha, buffer, codigo) != NULL) {
24         if (checa_vazia(linha, buffer) == 1) {
25             continue;
26         } else if (checa_comentario(linha, buffer, &flag_comentario) == 1) {
27             continue;
28         } else {
29             linhas_codigo++;
30         }
31     }
32     return linhas_codigo;
33 }
```

2.4.1.4 tamanho_linha()

```
int tamanho_linha (
              char * linha,
              const int buffer )
```

Percorre a linha fornecida até encontrar um caracter de quebra de linha, mantendo a contagem dos caracteres percorridos.

Retorna o tamanho da linha fornecida.

```
121                                                 {
122     int tamanho = 0;
123
124     while (linha[tamanho] != '\n' && tamanho < buffer) {
125         tamanho++;
126     }
127     return tamanho;
128 }
```

2.5 Referência do Arquivo conta_linhas/testes/conta_linhas.h

```
#include <stdio.h>
#include <stdbool.h>
```

Funções

- int conta_linhas_codigo (FILE *codigo)
- int checa_vazia (char *linha, const int buffer)
- int checa_comentario (char *linha, const int buffer, bool *flag_comentario)
- int tamanho_linha (char *linha, const int buffer)

2.5.1 Funções

2.5.1.1 checa_comentario()

```
int checa_comentario (
             char * linha,
             const int buffer,
             bool * flag_comentario )
```

Faz uso da flag_comentario para distinguir entre comentário de linha única e bloco de comentário.

Caso não esteja em bloco de comentário, pula todo caracter "whitespace" e analisa os dois primeiros válidos, para determinar se há um início de bloco de comentário nessa linha.

Caso esteja em bloco de comentário, analisa todos os caracteres da linha em pares, verificando se há ocorrência de término de bloco de comentário.

Retornos:
0, se linha não for de comentário;
1, se linha for de algum tipo de comentário.

Valores da flag_comentario:
0 -> não esta em bloco de comentário atualmente;
1 -> está em bloco de comentário.

```
78                                                                              {
79     int tamanho = tamanho_linha(linha, buffer);
80     int contador = 0;
81
82     switch (*flag_comentario) {
83         case 0:
84             while (isspace(linha[contador]) && contador < tamanho) {
85                 contador++;
86             }
87
88             if (contador == tamanho || contador+1 == tamanho) {
89                 return 0;
90             } else if (linha[contador] == '/') {
91                 switch (linha[contador+1]) {
92                     case '*':
93                         *flag_comentario = 1;
94                         return 1;
95                    case '/':
96                         return 1;
97                     default:
98                         return 0;
99                 }
100             } else {
101                 return 0;
102             }
103
104         case 1:
105             for (contador = 0; contador+1 < tamanho; contador++) {
106                 if (linha[contador] == '*' && linha[contador+1] == '/') {
107                     *flag_comentario = 0;
108                 }
109             }
110             return 1;
111     }
112 }
```

2.5.1.2 checa_vazia()

```
int checa_vazia (
            char * linha,
            const int buffer )
```

Percorre o array que contém a linha a ser analizada e checa se há algum caracter que não seja classificado como "whitespace".

Retornos:
0, se linha contiver caracter não-"whitespace";
1, se linha for vazia.

```
44                                                  {
45     int tamanho = tamanho_linha(linha, buffer);
46
47     for (int i = 0; i < tamanho; i++) {
48         if (!isspace(linha[i])) {
49             return 0;
50         }
51     }
52
53     return 1;
54 }
```

2.5.1.3 conta_linhas_codigo()

```
int conta_linhas_codigo (
            FILE * codigo )
```

Lê o arquivo de código fornecido linha por linha e, para cada linha, checa se é uma linha vazia ou se é uma linha de comentário.

Faz essas ações por meio de outras funções: checa_vazia e checa_comentario.

Retorna o número de linhas válidas de código.

```
17                                      {
18     const int buffer = 100;
19     int linhas_codigo = 0;
20     bool flag_comentario = 0;
21     char linha[buffer];
22
23     while (fgets(linha, buffer, codigo) != NULL) {
24         if (checa_vazia(linha, buffer) == 1) {
25             continue;
26         } else if (checa_comentario(linha, buffer, &flag_comentario) == 1) {
27             continue;
28         } else {
29             linhas_codigo++;
30         }
31     }
32     return linhas_codigo;
33 }
```

2.5.1.4 tamanho_linha()

```
int tamanho_linha (
            char * linha,
            const int buffer )
```

Percorre a linha fornecida até encontrar um caracter de quebra de linha, mantendo a contagem dos caracteres percorridos.

Retorna o tamanho da linha fornecida.

```
121                                                 {
122     int tamanho = 0;
123
124     while (linha[tamanho] != '\n' && tamanho < buffer) {
125         tamanho++;
126     }
127     return tamanho;
128 }
```

2.6 Referência do Arquivo conta_linhas/testes/testes.cpp

```
#include "conta_linhas.c"
#include <gtest/gtest.h>
```

Funções

- TEST (tamanho_linha, linha_vazia)
- TEST (tamanho_linha, linha_normal)
- TEST (checa_vazia, linha_vazia)
- TEST (checa_vazia, linha_codigo)
- TEST (checa_vazia, linha_comentario)
- TEST (checa_comentario, linha_vazia)
- TEST (checa_comentario, linha_codigo)
- TEST (checa_comentario, linha_comentario)
- TEST (conta_linhas_codigo, teste_geral_1)
- TEST (conta_linhas_codigo, teste_geral_2)
- TEST (conta_linhas_codigo, teste_geral_3)
- int main (int argc, char **argv)

2.6.1 Funções

2.6.1.1 main()

```
int main (
            int argc,
            char ** argv )
```

```
187                                 {
188     testing::InitGoogleTest(&argc, argv);
189     return RUN_ALL_TESTS();
190 }
```

2.6.1.2 TEST() [1/11]

```
TEST (
              tamanho_linha ,
              linha_vazia  )
```

Avalia capacidade da função de retornar zero quando fornecida uma linha vazia.

```
9                                  {
10     FILE* texto = fopen("linha_vazia.txt", "r");
11     char linha[35];
12
13     fgets(linha, 35, texto);
14     ASSERT_EQ(0, tamanho_linha(linha, 35));
15 }
```

2.6.1.3 TEST() [2/11]

```
TEST (
              tamanho_linha ,
              linha_normal  )
```

Avalia a capacidade da função de retornar o número correto de caracteres, dada uma linha válida de código.

```
22                                    {
23     FILE* texto = fopen("linha_codigo.txt", "r");
24     char linha[35];
25
26     fgets(linha, 35, texto);
27     ASSERT_EQ(18, tamanho_linha(linha, 35));
28 }
```

2.6.1.4 TEST() [3/11]

```
TEST (
              checa_vazia ,
              linha_vazia  )
```

Avalia capacidade da função retornar 1 quando fornecida uma linha vazia.

```
34                                 {
35     FILE* texto = fopen("linha_vazia.txt", "r");
36     char linha[35];
37
38     fgets(linha, 35, texto);
39     ASSERT_EQ(1, checa_vazia(linha, 35));
40
41     fclose(texto);
42 }
```

2.6.1.5 TEST() [4/11]

```
TEST (
              checa_vazia ,
              linha_codigo  )
```

Avalia capacidade da função retornar zero quando fornecida uma linha válida de código.

```
49                                  {
50     FILE* texto = fopen("linha_codigo.txt", "r");
51     char linha[35];
52
53     fgets(linha, 35, texto);
54     ASSERT_EQ(0, checa_vazia(linha, 35));
55
56     fgets(linha, 35, texto);
57     fgets(linha, 35, texto);
58     ASSERT_EQ(0, checa_vazia(linha, 35));
59
60     fclose(texto);
61 }
```

2.6.1.6 TEST() [5/11]

```
TEST (
              checa_vazia ,
              linha_comentario  )
```

Avalia capacidade da função retornar zero quando fornecida uma linha válida de comentário.

```
68                                      {
69     FILE* texto = fopen("linha_comentario.txt", "r");
70     char linha[35];
71
72     fgets(linha, 35, texto);
73     ASSERT_EQ(0, checa_vazia(linha, 35));
74
75     fgets(linha, 35, texto);
76     fgets(linha, 35, texto);
77     ASSERT_EQ(0, checa_vazia(linha, 35));
78
79     fclose(texto);
80 }
```

2.6.1.7 TEST() [6/11]

```
TEST (
              checa_comentario ,
              linha_vazia  )
```

Avalia capacidade da função retornar zero quando fornecida uma linha vazia.

```
86                                      {
87     FILE* texto = fopen("linha_vazia.txt", "r");
88     char linha[35];
89     bool flag = 0;
90
91     fgets(linha, 35, texto);
92     ASSERT_EQ(0, checa_comentario(linha, 35, &flag));
93
94     fclose(texto);
95 }
```

2.6.1.8 TEST() [7/11]

```
TEST (
              checa_comentario ,
              linha_codigo  )
```

Avalia capacidade da função retornar zero quando fornecida uma linha válida de código.

```
102                                     {
103     FILE* texto = fopen("linha_codigo.txt", "r");
104     char linha[35];
105     bool flag = 0;
106
107     fgets(linha, 35, texto);
108     ASSERT_EQ(0, checa_comentario(linha, 35, &flag));
109
110     fgets(linha, 35, texto);
111     fgets(linha, 35, texto);
112     ASSERT_EQ(0, checa_comentario(linha, 35, &flag));
113
114     fclose(texto);
115 }
```

2.6.1.9 TEST() [8/11]

```
TEST (
              checa_comentario ,
              linha_comentario  )
```

Avalia capacidade da função retornar 1 quando fornecida uma linha de código, enquanto retorna zero para linhas vazias.

```
122                                         {
123     FILE* texto = fopen("linha_comentario.txt", "r");
124     char linha[35];
125     bool flag = 0;
126
127     fgets(linha, 35, texto);    //linha 1
128     ASSERT_EQ(1, checa_comentario(linha, 35, &flag));
129
130     fgets(linha, 35, texto);    //linha 2
131     ASSERT_EQ(0, checa_comentario(linha, 35, &flag));
132
133     fgets(linha, 35, texto);    //linha 3
134     ASSERT_EQ(1, checa_comentario(linha, 35, &flag));
135
136     fgets(linha, 35, texto);    //linha 4
137     ASSERT_EQ(1, checa_comentario(linha, 35, &flag));
138
139     fgets(linha, 35, texto);    //linha 5
140     ASSERT_EQ(1, checa_comentario(linha, 35, &flag));
141
142     fgets(linha, 35, texto);    //linha 6
143     ASSERT_EQ(1, checa_comentario(linha, 35, &flag));
144
145     fgets(linha, 35, texto);    //linha 7
146     ASSERT_EQ(1, checa_comentario(linha, 35, &flag));
147
148     fgets(linha, 35, texto);    //linha 8
149     ASSERT_EQ(0, checa_comentario(linha, 35, &flag));
150
151     fclose(texto);
152 }
```

2.6.1.10 TEST() [9/11]

```
TEST (
              conta_linhas_codigo ,
              teste_geral_1  )
```

Avalia capacidade da função retornar o número correto de linhas de código no arquivo fornecido.

```
159                                     {
160     FILE* codigo = fopen("amostra1.cpp", "r");
161     ASSERT_EQ(12, conta_linhas_codigo(codigo));
162     //ignorar linhas vazias e comentarios
163 }
```

2.6.1.11 TEST() [10/11]

```
TEST (
              conta_linhas_codigo ,
              teste_geral_2  )
```

Avalia capacidade da função retornar o número correto de linhas de código no arquivo fornecido.

```
170                                     {
171     FILE* codigo = fopen("amostra2.cpp", "r");
172     ASSERT_EQ(16, conta_linhas_codigo(codigo));
173     //ignorar linhas vazias e comentarios
174 }
```

2.6.1.12 TEST() [11/11]

```
TEST (
              conta_linhas_codigo ,
              teste_geral_3  )
```

Avalia capacidade da função retornar o número correto de linhas de código no arquivo fornecido.

```
181                                     {
182     FILE* codigo = fopen("amostra3.cpp", "r");
183     ASSERT_EQ(11, conta_linhas_codigo(codigo));
184     //ignorar linhas vazias e comentarios
185 }
```

Índice Remissivo