Universidade Anhanguera-Uniderp - Unidade Vila Mariana

Marcel dos Santos Gubany – R.A. 7117508501

Plínio Balduino – R.A. 6950459640

Roberta Vieira Justino Balduino – R.A. 6950459481

ATPS – PROGRAMAÇÃO ESTRUTURADA 2

Desenvolvimento de um pequeno aplicativo de linha de comando – Console

São Paulo/SP

04/2014

Índice

1 Apresentação 3

2 Códigos fonte 3

2.1 Descrição 3

2.2 main.c 4

2.3 ccount.c 5

2.4 ccount.h 6

3 Arquivos de teste 8

3.1 teste1.txt 8

3.2 teste2.txt 8

3.3 teste3.txt 8

4 Demonstração de uso 9

4.1 Tela de ajuda 9

4.2 Lendo um arquivo 9

4.3 Silenciando a saída 9

# 1 Apresentação

O objetivo desta atividade é apresentar uma aplicação escrita na linguagem C que, dado um arquivo texto contendo um código fonte, também em linguagem C, informe ao usuário a quantidade de linhas com código, de linhas com comentário e de linhas em branco.

A aplicação deve ser ter uma interface de linha de comando, para que o usuário possa informar as configurações e parâmetros através do Prompt de Comandos do Windows.

Foi utilizado o compilador MinGW, que vem com o ambiente de desenvolvimento DevC++, para efetuar o desenvolvimento e os testes. O código é compatível com o compilador GCC, desde que utilizada a flag –std=c99.

# 2 Códigos fonte

## 2.1 Descrição

A aplicação é composta por três arquivos: *main.c*, *ccount.c* e *ccount.h*.

O arquivo *main.c* contém a manipulação dos parâmetros recebidos através da linha de comando e, uma vez que estes estejam corretos, invoca a função cc\_run, que é quem cuida dos cálculos de linhas e comentários.

Já o arquivo *ccount.c* contém os códigos das funções cc\_run, e também da função cc\_show\_options, responsável por exibir as opções de uso para o usuário.

A função cc\_count tem sua lista de parâmetros organizada de uma forma que primeiro sejam recebidos os parâmetros de entrada e, em seguida, sejam informados os parâmetros de saída, por onde serão devolvidos os resultados da função. É uma boa prática fazer com que cada parâmetro seja de entrada ou de saída, nunca os dois ao mesmo tempo.

Finalmente, o arquivo ccount.h contém, além dos protótipos das funções contidas no arquivo ccount.c, as definições dos tipos BOOL, como forma de deixar claro no código que estamos lidando com um valor lógico/booleano; e ptrINT, ptrCHAR e ptrFILE, que são formas de abstrair os ponteiros para os tipos int, char e File, respectivamente. Como efeito colateral, essa nomenclatura faz com que fique claro que estamos lidando com um parâmetro de saída, uma vez que um argumento que se utilize de ponteiros é a forma comum de passarmos um valor por referência.

Neste arquivo são definidos também os valores de TRUE e FALSE, tornando explícito em sua declaração que são tipos BOOL e que TRUE é o complemento de FALSE.

## 2.2 main.c

#include "include/ccount.h"

void pause() {

system("PAUSE");

}

int main(int argc, ptrCHAR argv[]) {

int lines = 0;

int commented\_lines = 0;

int empty\_lines = 0;

ptrCHAR filename = NULL;

BOOL silent = FALSE;

BOOL comment = FALSE;

BOOL help = FALSE;

int p;

setlocale(LC\_ALL, "portuguese");

for (p = 1; p < argc; p++) {

if ((strcmp(argv[p], PARAM\_HELP\_LONG) == 0)

|| (strcmp(argv[p], PARAM\_HELP\_SHORT) == 0)) {

help = TRUE;

break;

} else if ((strcmp(argv[p], PARAM\_COMMENT\_SHORT) == 0)

|| (strcmp(argv[p], PARAM\_COMMENT\_LONG) == 0)) {

comment = TRUE;

} else if ((strcmp(argv[p], PARAM\_SILENT\_SHORT) == 0)

|| (strcmp(argv[p], PARAM\_SILENT\_LONG) == 0)) {

silent = TRUE;

} else {

filename = argv[p];

}

}

if (help) {

cc\_show\_options();

} else {

if (filename) {

FILE\* file = fopen(filename, "r");

if (file) {

cc\_run(file, comment, silent,

&lines, &commented\_lines, &empty\_lines);

if (!silent) {

printf("---\n");

printf("Lendo arquivo %s\n", filename);

printf(" Linhas de codigo: %d\n", lines);

printf(" Linhas em branco: %d\n", empty\_lines);

if (commented\_lines) {

printf(" Linhas comentadas: %d\n", commented\_lines);

}

}

fclose(file);

} else {

printf("Arquivo %s nao encontrado\n", filename);

}

} else {

printf("Informe o arquivo a ser lido\n");

}

pause();

}

return 0;

}

## 2.3 ccount.c

#include "include/ccount.h"

void cc\_run(prtFILE file, BOOL comment, BOOL silent,

ptrINT lines, ptrINT commented\_lines, ptrINT empty\_lines) {

char c;

BOOL empty = TRUE;

BOOL comment\_candidate = FALSE;

BOOL end\_comment\_candidate = FALSE;

BOOL one\_line\_comment = FALSE;

BOOL multi\_line\_comment = FALSE;

BOOL already\_counted = FALSE;

if ((c = fgetc(file)) != EOF) {

(\*lines)++;

}

while(c != EOF) {

if(c == '\n') {

if(empty) {

(\*empty\_lines)++;

} else if(one\_line\_comment || multi\_line\_comment) {

one\_line\_comment = FALSE;

(\*commented\_lines)++;

} else if (!already\_counted){

(\*lines)++;

} else {

already\_counted = FALSE;

}

empty = TRUE;

comment\_candidate = FALSE;

} else if (c != ' ' || c != '\r') {

empty = FALSE;

if(c == '/') {

if(multi\_line\_comment && end\_comment\_candidate) {

multi\_line\_comment = FALSE;

end\_comment\_candidate = FALSE;

(\*commented\_lines)++;

already\_counted = TRUE;

} else if (!comment\_candidate) {

comment\_candidate = TRUE;

} else {

comment\_candidate = FALSE;

one\_line\_comment = TRUE;

}

} else if (comment\_candidate && c == '\*') {

comment\_candidate = FALSE;

multi\_line\_comment = TRUE;

} else if (multi\_line\_comment && c == '\*') {

end\_comment\_candidate = TRUE;

} else {

end\_comment\_candidate = FALSE;

}

}

c = fgetc(file);

}

}

void cc\_show\_options() {

printf("\nCodeCount: Contador de linhas de um codigo fonte em C/C++\n");

printf(" ATPS de Programacao Estruturada 2\n");

printf(" Uso:\n");

printf(" CodeCount [opcoes] arquivo.c[pp]\n");

printf(" Opcoes:\n");

printf(" -h --help : Exibe esta tela\n");

printf(" -c --comment: Exibe a contagem de linhas comentadas\n");

printf(" -s --silent : Nao exibe informacoes na tela\n");

}

## 2.4 ccount.h

#include <locale.h>

#include <stddef.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef unsigned char BOOL;

typedef int\* ptrINT;

typedef char\* ptrCHAR;

typedef FILE\* prtFILE;

#define FALSE (BOOL)0

#define TRUE (BOOL)!FALSE

#define PARAM\_HELP\_SHORT "-h"

#define PARAM\_HELP\_LONG "--help"

#define PARAM\_SILENT\_SHORT "-s"

#define PARAM\_SILENT\_LONG "--silent"

#define PARAM\_COMMENT\_SHORT "-c"

#define PARAM\_COMMENT\_LONG "--comment"

void cc\_run(prtFILE file, BOOL comment, BOOL silent,

ptrINT lines, ptrINT commented\_lines, ptrINT empty\_lines);

void cc\_show\_options();

# 3 Arquivos de teste

## 3.1 teste1.txt

Arquivo

Segunda linha

// comentario de uma linha

/\*

linha comentada

outra linha comentada

\*/

// e aqui um comentario que nao comeca na primeira coluna

fim

em branco: 2 linhas

comentadas: 6 linhas

linhas de codigo: 7

total: 15 linhas

## 3.2 teste2.txt

// Este eh um comentario de uma linha so.

for (i = 0; i < 10; i++)

{

/\* Este eh um comentario de duas linhas.

O termino eh ocorre na proxima linha.

\*/

printf("Valor de i %d", i); //Um comentario aqui nao deve ser computado.

}

## 3.3 teste3.txt

int main(int argc, char \*argv[]) {

int i, silent;

int nLines, nLinesComment, nLinesEmpty, count;

silent = 0;

nLines = 0;

// um comentário

printf("Code Count - Programmer's Tool\n");

printf(" Version 1.1 - 2009\n\n");

if (!silent)

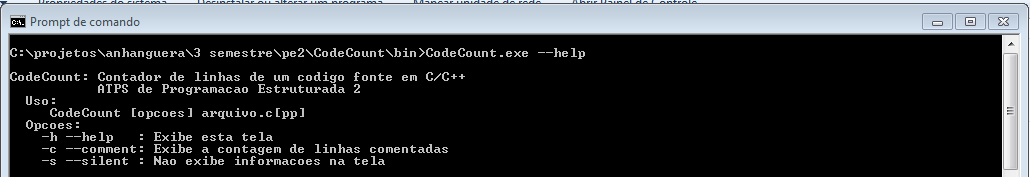
printf(" counting for \"%s\"...\n", argv[1]);

ccRun(argv[1], &nLines, &nLinesComment, &nLinesEmpty, noCom, silent);

# 4 Demonstração de uso

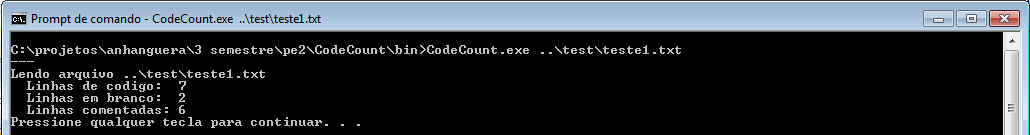
## 4.1 Tela de ajuda

Utilize a opção –h para exibir a tela abaixo



## 4.2 Lendo um arquivo

Informando o nome do arquivo como primeiro parâmetro, teremos o resultado como na tela abaixo



## 4.3 Silenciando a saída

Ao informarmos a opção –s, silenciamos a saída da aplicação.

