Guião aula05 - Gil Guedes 125031

1 - a - O programa verifica se há pelo menos 2 argumentos quando é chamado. Caso não aconteça imprime uma mensagem de erro, por outro lado, caso tenha tudo direitinho vai imprimir o número do argumento com este à frente.

b -

```c
#include <stdio.h>
#include <stdlib.h>

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

   if(argc != 3){
       printf("Erro: O programa deve receber exatamente dois argumentos!\n");
       printf("Uso: %s <arg1> <arg2>\n", argv[0]);
       return EXIT_FAILURE;
   }

   for(i = 0 ; i < argc ; i++)
   {
       printf("Argument %02d: \"%s\"\n", i, argv[i]);
   }

   return EXIT_SUCCESS;
}
```

c -

```c
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <math.h>

int main(int argc, char *argv[])
{
   double num1, num2, resultado;
   char *endptr1, *endptr2;

   if(argc != 4){
       printf("Uso: %s <numero1> <operacao> <numero2>\n", argv[0]);
       printf("Operações disponíveis: +, -, x, /, p (potência)\n");
       printf("Números podem ser inteiros ou decimais (ex: 3.14, -2.5)\n");
       return EXIT_FAILURE;
   }

   errno = 0;
   num1 = strtod(argv[1], &endptr1);
   num2 = strtod(argv[3], &endptr2);

   if (errno || endptr1 == argv[1] || *endptr1 || endptr2 == argv[3] || *endptr2) {
       printf("Erro: argumento inválido!\n");
       return EXIT_FAILURE;
```

```
    }

    switch (argv[2][0])
    {
    case '+':
        resultado = num1 + num2;
        printf("%.6g + %.6g = %.6g\n", num1, num2, resultado);
        break;

    case '-':
        resultado = num1 - num2;
        printf("%.6g - %.6g = %.6g\n", num1, num2, resultado);
        break;

    case 'x':
        resultado = num1 * num2;
        printf("%.6g x %.6g = %.6g\n", num1, num2, resultado);
        break;

    case '/':
        if(num2 == 0.0){
            printf("Erro: Divisão por zero não é permitida!\n");
            return EXIT_FAILURE;
        }
        resultado = num1 / num2;
        printf("%.6g / %.6g = %.6g\n", num1, num2, resultado);
        break;

    case 'p':
        if(num2 < 0 && num1 == 0.0){
            printf("Erro: 0 elevado a potência negativa é indefinido!\n");
            return EXIT_FAILURE;
        }
        resultado = pow(num1, num2);
        printf("%.6g ^ %.6g = %.6g\n", num1, num2, resultado);
        break;

    default:
        printf("Operação inválida: %s\n", argv[2]);
        printf("Operações disponíveis: +, -, x, /, p (potência)\n");
        return EXIT_FAILURE;
    }

    return EXIT_SUCCESS;
}
```

d - O programa com “atof” tem problemas, pois não deteta erros, exemplo se o argumento não for numérico "abc", devolve 0.0 sem indicar erro; também não permite distinguir entre erro de conversão e número válido igual a 0.

e - Como o ‘*’ é um caractere especial de expansão, ao o usar ele vai expandir para todos os ficheiros do diretório atual antes da calculadora rodar, mudando o input, fazendo com que dê erro

2 - a - O programa obtém usuário da variável USER e conta caracteres dos argumentos

c -

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[])
{
   int i, total_length = 0;
   char *joined_text;

   if (argc < 2) {
       printf("Uso: %s <palavra1> <palavra2> ... <palavraN>\n", argv[0]);
       return EXIT_FAILURE;
   }

   for (i = 1; i < argc; i++) {
       total_length += strlen(argv[i]);
       if (i < argc - 1) {
           total_length += 1;
       }
   }
   total_length += 1;

   joined_text = (char*)malloc(total_length * sizeof(char));
   if (joined_text == NULL) {
       printf("Erro: Não foi possível alocar memória!\n");
       return EXIT_FAILURE;
   }

   strcpy(joined_text, "");

   for (i = 1; i < argc; i++) {
       strcat(joined_text, argv[i]);
       if (i < argc - 1) {
           strcat(joined_text, " ");
       }
   }

   printf("Frase completa: %s\n", joined_text);
   printf("Total de caracteres: %d\n", (int)strlen(joined_text));

   free(joined_text);
   return EXIT_SUCCESS;
}
```

d -

```c
    int first_valid = 1;
   for (i = 1; i < argc; i++) {
       if (strlen(argv[i]) > 0 && isalpha(argv[i][0])) {
           if (!first_valid) {
               strcat(joined_text, " ");
           }
           strcat(joined_text, argv[i]);
           first_valid = 0;}}
```

4 - a -

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

int compare_ascending(const void *a, const void *b) {
   return strcmp(*(char**)a, *(char**)b);
}

int compare_descending(const void *a, const void *b) {
   return strcmp(*(char**)b, *(char**)a);
}

int compare_ascending_ignore_case(const void *a, const void *b) {
   return strcasecmp(*(char**)a, *(char**)b);
}

int compare_descending_ignore_case(const void *a, const void *b) {
   return strcasecmp(*(char**)b, *(char**)a);
}

int main(int argc, char *argv[])
{
   int i, valid_count = 0;
   char **valid_words;
   char *sort_order;
   char *ignore_case_env;
   int ignore_case = 0;
   int ascending = 1;

   if (argc < 2) {
       printf("Uso: %s <palavra1> <palavra2> ... <palavraN>\n", argv[0]);
       printf("Nota: Apenas argumentos que começam com letra serão ordenados.\n");
       printf("Variáveis de ambiente:\n");
       printf("  SORTORDER: 'asc' ou 'desc' (padrão: asc)\n");
       printf("    IGNORECASE: 'yes' para ignorar maiúsculas/minúsculas (padrão:
no)\n");
       return EXIT_FAILURE;
   }

   sort_order = getenv("SORTORDER");
   if (sort_order != NULL) {
       if (strcmp(sort_order, "desc") == 0 || strcmp(sort_order, "DESC") == 0) {
           ascending = 0;
       } else if (strcmp(sort_order, "asc") == 0 || strcmp(sort_order, "ASC") == 0)
{
           ascending = 1;
       } else {
           printf("Aviso: SORTORDER='%s' inválido. Usando ordenação crescente.\n",
                  sort_order);
           printf("Valores válidos: 'asc', 'desc', 'ASC', 'DESC'\n\n");
       }
   }

   ignore_case_env = getenv("IGNORECASE");
   if (ignore_case_env != NULL) {
       if (strcmp(ignore_case_env, "yes") == 0 ||
```

```
            strcmp(ignore_case_env, "YES") == 0 ||
            strcmp(ignore_case_env, "1") == 0) {
            ignore_case = 1;
        }
    }

    for (i = 1; i < argc; i++) {
        if (strlen(argv[i]) > 0 && isalpha(argv[i][0])) {
            valid_count++;
        }
    }

    if (valid_count == 0) {
          printf("Nenhum argumento válido encontrado (argumentos devem começar com
letra).\n");
        return EXIT_SUCCESS;
    }

    valid_words = (char**)malloc(valid_count * sizeof(char*));
    if (valid_words == NULL) {
        printf("Erro: Não foi possível alocar memória!\n");
        return EXIT_FAILURE;
    }

    int index = 0;
    for (i = 1; i < argc; i++) {
        if (strlen(argv[i]) > 0 && isalpha(argv[i][0])) {
            valid_words[index] = argv[i];
            index++;
        }
    }

    printf("=== CONFIGURAÇÃO DE ORDENAÇÃO ===\n");
    printf("Ordem: %s\n", ascending ? "Crescente" : "Decrescente");
    printf("Ignorar maiúsculas/minúsculas: %s\n", ignore_case ? "Sim" : "Não");
    printf("Argumentos válidos encontrados: %d\n\n", valid_count);

    if (ignore_case) {
        if (ascending) {
                                    qsort(valid_words,   valid_count,   sizeof(char*),
compare_ascending_ignore_case);
        } else {
                                    qsort(valid_words,   valid_count,   sizeof(char*),
compare_descending_ignore_case);
        }
    } else {
        if (ascending) {
            qsort(valid_words, valid_count, sizeof(char*), compare_ascending);
        } else {
            qsort(valid_words, valid_count, sizeof(char*), compare_descending);
        }
    }

    printf("=== ARGUMENTOS PROCESSADOS ===\n");
    for (i = 1; i < argc; i++) {
        if (strlen(argv[i]) > 0 && isalpha(argv[i][0])) {
            printf("  ✓ '%s' (incluído)\n", argv[i]);
        } else {
```

```
            printf("  ✗ '%s' (ignorado - não inicia com letra)\n", argv[i]);
        }
    }

    printf("\n=== PALAVRAS ORDENADAS ===\n");
    for (i = 0; i < valid_count; i++) {
        printf("%d. %s\n", i + 1, valid_words[i]);
    }

    free(valid_words);
    return EXIT_SUCCESS;
}
```