AP1 – Introdução à Programação C/C++
Nome: Nathan Andrade dos Santos Lobo
DRE: 120082390

1. O algoritmo foi escrito pensado em fazer uma verificação do scanner do teclado do usuário verificando se o que ele digitou condiz com a dada opção do problema. O cálculo da porcentagem é feito a partir do total de votos e o total é calculado a cada laço. O laço é infinito pois é possível quebrá-lo a fim de evitar contar mais uma vez o total, se o usuário digitar 0 como condição de parada.

2. **tamanho():** Faço um for de i até o tamanho pré-alocado onde eu vou somando a cada iteração e quebro o laço assim que chegar na última posição da palavra que é o "**\0"**. No final, retorno o tamanho – 1 pois recebo a palavra do usuário pela função **fgets()**, a qual adiciona +1 ao tamanho da palavra por conta do **"\n"** que a função mesmo gera.

   copia(): Faço um for de i até o tamanho da palavra origem e recebo cada caractere de origem em destino.

   concatena(): Utilizei uma variável global para trabalhar na função "concatena" de modo que essa variável controle o resultado que vai vir do tamanho de i (variável iteradora de 0 até o tamanho do destino), recebendo cada caractere de destino. A partir disso, faço concatenar da posição de i + destino, para continuar adicionando novos caracteres que agora vêm da palavra origem.

   inverte(): Itero de i até o tamanho da palavra destino e faço a palavra destino na posição i, receber a origem da posição do (tamanho da origem – i – 1). O -1 é para manter o índice dentro dos índices finais da palavra.

   compara(): Faço uma simples verificação se o tamanho das duas palavras são o mesmo, e se não for, verifico se é menor ou maior. Caso sejam iguais, 0, do contrário, 1.

   Na função principal faço as chamadas pedindo o input do usuário pelo **fgets()** e faço a chamada de todas as funções.

3. O palíndromo recursivo tem como condição de parada se o tamanho da variável **i** for o mesmo que o tamanho da variável **tam – 1**, então a palavra é um palíndromo.
   A garantia por ser "iguais" é feita antes dessa pausa, verificando as primeiras posições com as últimas posições da string, e para isso, utilizo uma variável auxiliar e vou somando-a a cada chamada feita pela recursividade.

   Na função principal, além de ler o input do usuário novamente pelo **fgets()** faço um for apenas como um "trim", para remover acentuações, pontuações e espaços em branco utilizando a Tabela ASCII como modelo. Se a palavra que o usuário digitou

não conter um caractere que esteja dentro de **a-z** ou **A-Z**, então o for continua para o próximo laço. Crio uma variável **j** para servir como tamanho da nova palavra.

4. Peço para o usuário preencher o Vetor A e o Vetor B, de tamanho já estabelecido, e faço um for para cada iteração ir acumulando a variável **“soma_vetor_a”** que serve como somatório de todos os elementos do vetor. O mesmo faço para o Vetor B com a variável **“soma_vetor_b”**.
   Para o Vetor C, faço um for que vai de i até o tamanho do vetor e guardo na posição i do vetor a soma do Vetor A **na posição i** com o Vetor B **na mesma posição**.
   Para o Vetor D, repito o que fiz para o C, entretanto, subtraio do Vetor B com o Vetor A e guardo na posição i.
   No caso do produto escalar, eu faço o somatório do produto de **VetorA[i]** com o **VetorB[i]**.

5.
   a) Utilizo uma função de soma recursiva que vai somando até chegar à posição inicial do vetor.

   b) Faço uma checagem se o vetor recebido na posição é ímpar ou não. Se for ímpar, retorno 1 para fazer o cálculo dos índices na função **main()**. Na função main, verifico de 0 até o tamanho do vetor, se o número é ímpar e caso seja, salvo em uma variável **index** e a incremento a cada checagem do laço.

   c) Utilizo a mesma ideia da função de soma recursiva para fazer o produtório de modo recursivo. A única diferença é que, só executo o cálculo se o número for de fato par e para isso checamos com a aritmética modular. Se o número não for par, diminui-se uma posição do vetor, visto que a condição de parada é a posição inicial.

   d) Para somar o triplo dos elementos de índice ímpar, eu faço a checagem se o elemento que está na posição dada é primo e caso seja, faço a somatória e vou triplicando a cada laço na chamada da função. Na **main()** apenas somo os resultados com o laço mencionado.

   e) Uma simples função que divide cada posição por 4.0 e retorna o novo valor dividido. Na função principal executo um laço para fazer a divisão de cada posição até o fim do vetor.

   f) Checo se o número que está em uma dada posição do vetor é maior que o maior número e se for, o maior passa a ser este número. Inicializo o maior na função main() sendo 0, para sempre ter com o que comparar.

   g) Repito o mesmo procedimento do maior, entretanto, inicializo o menor na main() com um número grande de modo que ele seja sempre comparado dentro da função, para que ele passe a ser o menor número do vetor.

h) Para calcular a média dos elementos faço apenas a chamada para a função de soma, que faz o somatório de todos os elementos do vetor e divido pelo tamanho do vetor.

i) Para calcular elementos que são menores do que a média, tomo uma variável chamada média que recebe a média e faço um laço de 0 até o tamanho do vetor, para checar se a média é maior do que o número que está na posição i e se for, mostro ao usuário que o número da posição i é menor que a média.

j) Executo a mesma lógica da função que calcula os elementos menores do que a média, com a diferença de que, se o elemento na posição i do vetor for maior que a média, então eu mostro ao usuário o resultado da variável i que está sendo incrementada a cada laço.

k) Utilizo um algoritmo baseado no famoso Bubble Sort, faço um for que vai do início até o fim do vetor, incrementando a cada passagem do laço, e faço um for para auxiliar a verificação do próximo elemento do vetor, indo de j recebendo 0 até tamanho – i – 1, de modo que não passe do final do vetor e sempre esteja dentro dos elementos existentes.

   No for auxiliar, faço uma verificação se o elemento que está na posição j é maior do que o próximo elemento e se for, salvo o elemento da posição j em uma variável, em seguida, transfiro o elemento da posição seguinte para a posição anterior e o elemento que estava salvo na variável guardo na próxima posição, que estaria “vazia” após passar o elemento para anterior.

Questão Bônus:

Para a questão bônus, sigo exatamente o design do algoritmo já escrito pelo professor, de modo que eu crio três variáveis x, y e direcao globalmente.

Crio seis funções (**avancar, voltar, girar_esquerda, girar_direita**) e sigo a lógica do plano cartesiano.

Faço a função principal, ligar trabalhar com o menu e toda a atualização das variáveis. Neste menu, o usuário recebe 5 instruções, como: Parar o Robô, Avançar o Robô, Voltar o Robô, Girar o Robô para direita, Girar o Robô para esquerda. Se o usuário para o robô, o programa consequentemente acaba.

Para as funções avancar e voltar, a mudança na posição é diretamente no y enquanto girar à esquerda diminui um na variável x e girar à direita aumenta um, também na variável x, seguindo a ideia dos pontos cardeais.