Identificação: Pedro Mesquita Maia – 2312664

Descrição: O objetivo do trabalho foi gerar duas listas de números decimais randomicamente, ordenar a primeira lista, e em seguinda criar uma terceira lista contedo os números que não são comuns entre as listas de números randomicos. Além disso, foi pedido para contar o tempo decorrido da operação de verificar e adicionar os números não comuns das listas randomicas.

Estrutura: Foram criadas cinco funções, além da main, para realizar o trabalho (randomNumber, populateList, sortList, findNumber e printList). Após a criação da três listas, foi chamada a função "populateList" e "randomNumber" para popular duas delas, em seguida foi chamada a função de "sortList" e depois a "findNumber". Por último foi utilizada a função "printList" para mostrar os valores.

Solução: Foram criadas duas constantes (MIN = 0 e MAX = 10000) usadas para definir o tamanho das listas, dos laços de repetição e para gerar os números aleatórios. Foram criadas cinco funções, além da main, para realizar o trabalho (randomNumber, populateList, sortList, findNumber e printList).
Após a criação da três listas, foi chamada a função populateList para popular duas delas.

A função populateList recebe um ponteiro para float e através de um laço for (usando as constantes) atribui para cada posição da lista o retorno da função randomNumber

A função randomNumber retorna um float e através da função rand gera um número inteiro. Para gerar números de ponto flutuante foi necessário fazer o casting para float da função, dividir esse valor pelo casting do RAND_MAX (constante do stdlib que define o valor máximo de retorno do rand) e multiplicar por MAX. A divisão realiza ela gera números aleatórios entre 0 e 1, já a multiplicação por MAX faz com que o valor máximo seja o próprio MAX.

Após as duas listas serem populas, uma delas é ordenada. A função sortList recebe um ponteiro para float e declara duas variáveis do tipo int que vão servir para percorrer os laços em seguida. No primeiro laço for (de MIN até MAX), é declarado uma variável inteira chamada exitFlag que é inicializada com 0 (Essa flag vai ser usada para sair do laço se o próximo valor não for maior que o valor atual, significando que toda a lista já está ordenada), ainda dentro do primeiro laço for vamos ter outro laço for que vai ser responsável por percorrer toda a listagem. O segundo laço for possui uma verificação que valida se o valor atual da lista é maior que o próximo valor da lista, caso seja eu salvo o valor atual em uma variável temporaria, atribuo a posição atual da lista o valor da próxima posição e a posição seguinte recebe o valor da variável temporária, além disso a flag recebe o valor de 1. Após o segundo laço, há um verificação que valida o valor da flag, caso ela seja 0 (toda a lista está ordenada) ela sai do laço for.

Após isso, na função main, eu declaro as estruturas (timeval) e variáveis necessárias para usar a função gettimeofday que registra momentos de tempo. A minha função findNumber está entre duas chamadas da função getimeofday passando como parâmetro o endereço da variável start e end respectivamente.

Entre as chamadas da função gettimeofday eu declaro uma variável inteira chamada thirdListSize que recebe o retorno da função findNumber. A função findNumber recebe três endereços de vetores, na fução é definido a variável "quantity" inteira que recebe 0 (ela vai ser usada como índice para atribuir o valor diferente na terceira lista e também para facilitar a hora de "printar" a terceira lista na função printList), após essa declaração é criado um loop for (usando as constantes) para percorrer 10.000 vezes e garantir que todos os valores foram comparados e declarado uma variável inteira "found" que

recebe 0 (vai ser usada para sair do segundo laço quando for encontrado o primeiro caso em que o número da posição da primeira lista é igual ao número da posição atual da segunda lista, fazendo com que não seja necessário percorrer toda a lista uma vez que já foi encontrado um caso). Após a variável “found”, é criado o segundo loop (usando as constantes) que vai ser usada para comparar os valores entre a primeira e a segunda lista. Dentro do segundo loop, há um if que verifica o valor atual da primeira lista (i) e o valor atual da segunda lista (j), caso o valor seja igual a variável “found” recebe 1 e faz um break (saindo do loop atual). Após o segundo loop há uma outra verificação que valida se o “found” é diferente de 0 (falso), caso seja, é adicionada na terceira lista, na posição quantity++ (para ir para o próximo índice da terceira lista) o valor atual da primeira lista (no caso, o valor que é diferente). Por fim, é retornado o valor de ”quantity”.

Na main, após salvar o valor de ”quantity” na variável inteira “thirdListSize” é pego o momento atual e salvo no endereço que a variável ”end” aponta. Em seguida é calculado o valor de segundo, microsegundos e tempo total descorrido usando o valor das variáveis “start” e “end” e, após operações, atribuindo a variável elapsed.

Após calcular o tempo de execução da função “findNumber”, é executada a função “printList” que é responsável por “printa” a lista passada com base na variável “thirdListSize”. Nessa função, basicamente é um laço for (usando MIN e thirdListSize) para printar cada valor que está na terceira lista e também mostra, depois do loop, a quantidade de números diferentes.

Em seguida a função de “print” também é mostrado o tempo gasto em segundos com base na variável “elapsed”.

Observações:

Por não conhecer a função gettimeofday eu fiquei um pouco perdido na hora de implementar, principalmente por pedir variáveis de um tipo específico, além disso tive que pesquisar sobre como acessar os valores salvos nas variáveis passadas. Um outro problema foi gerar os números decimais aleatórios, pois a função rand só retorna valores inteiros.
De modo geral eu achei a parte de comparação e ordenação tranquilas de fazer, pois não pediam nenhum raciocínio muito complexo ou que utilizasse funções não tão comuns para mim.

Para compilar e observar possíveis warning eu utilizei o seguinte comando: “gcc -Wall -o main main.c”. Como se tratava de um programa de números aleatórios, a única coisa que eu testei foram os tamanhos das listagem e funcionou com listagem de tamanhos 1000 e 100 (além do tamanho 10.000).