PROBLEMA SIMPLES 3

1 Analise Inicial

Foi pedido para se desenvolver uma função que recebe uma lista de inteiros e retorna uma lista apenas com os valores primos encontrados. Para resolver esse problema, decidiu-se usar um laço para verificar se cada um dos valores dentro da lista recebida possuiam divisores e rejeitar todos os que apresentassem, no mínimo, um divisor diferente de 1 ou dele mesmo.

2 Funções

O problema pede para fazer uma função que recebe uma lista de inteiros e retorna uma lista só com os valores primos encontrados. Para isso, fez-se uma função que retorna um ponteiro de inteiro, recebe um ponteiro de inteiro (a lista analisada) e o valor do tamanho da lista. Escolheu-se essa abordagem pois foi a melhor forma encontrada para evitar o acesso a posições inválidas no vetor, sem ter que alocar um espaço maior do que o necessário.

Primeiramente, foi feito o alocamento de memória para a composição da lista de primos com o mesmo tamanho da lista de entrada, já que esse é o tamanho máximo que a lista de primos pode assumir.

```
//Retorna um vetor preenchido com os primos presentes na lista. Caso nao haja
nenhum primo na lista, retorna NULL
int* filtra_primos(int *lista, int tam){ //Recebe uma lista e seu tamanho
  int *primos = NULL, ctd = 0, flag = 0, max_div = 0;

  primos = malloc(tam*sizeof(int)); //Aloca o maior tamanho possivel para a lista
final, que e o mesmo da lista de entrada
```

Figura 1: Criação e Alocamento de Espaço para a Lista de Primos

Então, fez-se um laço *for* para percorrer os valores da lista. Primeiro, uma variável de flag é abaixada e é verificado se o número observado é igual a 1. Como 1 não é primo, se esse caso for verdadeiro, a flag é levantada.

Após isso, guarda-se o valor da raiz quadrada do número observado para usá-la como limite da verificação em seguida. Adotou-se esse limite pois, se um número ”p”não é divisível por nenhum número menor que sua raiz, ele não será divisível por nenhum número maior que

ela. Isso se dá porque, se esse fosse o caso, deveria exitir outros dois valores ”a”e ”b”maiores que $\sqrt{p}$ tais que $a \cdot b = p$, o que é um absurdo.

Então, percorrem-se os números de 2 até a raiz do número observado e, caso seja encontrado um divisor, levanta-se a flag para sinalizar que o número não é primo e quebra-se o laço. Se o laço terminar com a flag abaixada, o número observado é primo.

```
  for(int i = 0; i < tam; i++){ //Percorre a lista
    flag = 0;
    if(lista[i] == 1) flag = 1; //Caso especial do numero 1
    max_div = (int) (sqrt(lista[i])); //O maior divisor primo de um numero será
sua raiz quadrada, logo nao e necessario verificar numeros apos ela
    for(int j = 2; j <= max_div; j++) //Percorre todos os numeros entre 2 e a raiz
quadrada do numero analisado
      if(lista[i] % j == 0){ //Se encontrar um divisor, levanta o flag e quebra o
laço
        flag = 1;
        break;
      }
    if(!flag) primos[ctd++] = lista[i]; //Se o flag nao estiver erquido, o numero
e primo
  }
```

Figura 2: Laço de Verificação de Números Primos

Com isso, após analisar cada termo da lista de entrada, terá-se uma lista com todos os primos encontrados. Vale lembrar que a variável ”ctd”foi usada para percorrer os índices da lista de primos e, no final, tem o valor do tamanho da lista de primos.

Finalmente, verifica-se se a lista de primos está vazia e, caso esteja, libera-se a memória alocada e retorna NULL. Senão, faz-se um realocamento da memória para reduzir o tamanho da lista de primos para apenas o valor guardado na variável ”ctd”, com o objetivo de evitar a ocupação de memória não utilizada. Após isso, retorna o ponteiro da lista de primos.

```
  if(ctd == 0){
    free(primos);
    return NULL; //Caso nao tenha nenhum primo, retorna NULL
  }

  primos = realloc(primos, ctd*sizeof(int)); //Reduz o tamnho alocado para a
quantidade de primos encontrada

  return primos; //Retorna a lista de primos
}
```

Figura 3: Retornos da Função