Universidade Veiga de Almeida

Análise e desenvolvimento de sistemas

Jonathan Da Silva Vieira

Estrutura de Repetição com For.
(Par ou Ímpar Computacional)

RIO DE JANEIRO - RJ 2025

RESUMO

No vasto cenário do desenvolvimento de software, a linguagem C se destaca por sua reconhecida eficiência e flexibilidade, características que a tornam uma ferramenta poderosa. Dentre seus pilares fundamentais, encontram-se as estruturas de repetição, mecanismos cruciais que possibilitam a execução iterativa de trechos de código sob condições específicas. O comando **for**, em particular, desempenha um papel essencial nesse contexto, sendo fundamental para automatizar tarefas repetitivas e para o processamento sistemático e eficaz de sequências de dados.

Nesse sentido, o presente trabalho se concentra na aplicação prática do comando **for** no desenvolvimento de um programa em linguagem C. O objetivo central é solucionar a tarefa de verificar a paridade de números inteiros dentro de um intervalo definido interativamente pelo usuário. Através da iteração proporcionada pela estrutura **for**, e com o auxílio de uma função dedicada à análise da divisibilidade por dois, o programa determinará e exibirá se cada número dentro do espectro estabelecido é par ou ímpar, demonstrando de forma clara o poder e a aplicabilidade das estruturas de repetição na resolução de problemas computacionais.

Este trabalho busca atender aos critérios de avaliação da disciplina, demonstrando conhecimento e domínio da estrutura de repetição **for** e da criação de funções em C para a resolução da situação problematizadora proposta.

A clareza na exposição das ideias e a argumentação sobre as escolhas de implementação do código serão elementos centrais deste trabalho, visando uma compreensão aprofundada do processo de desenvolvimento.

Arquitetura e Funcionamento do Programa

O programa desenvolvido em linguagem C tem como objetivo central a análise da paridade de números inteiros dentro de um intervalo estabelecido pelo usuário. A solução implementada inicia-se com uma interação amigável, na qual o usuário é convidado a fornecer os limites inferior e superior do intervalo desejado. Uma vez definidos esses limites, o programa emprega a robusta estrutura de repetição **`for`** para iterar sequencialmente por cada número compreendido nesse intervalo, incluindo os valores limites."

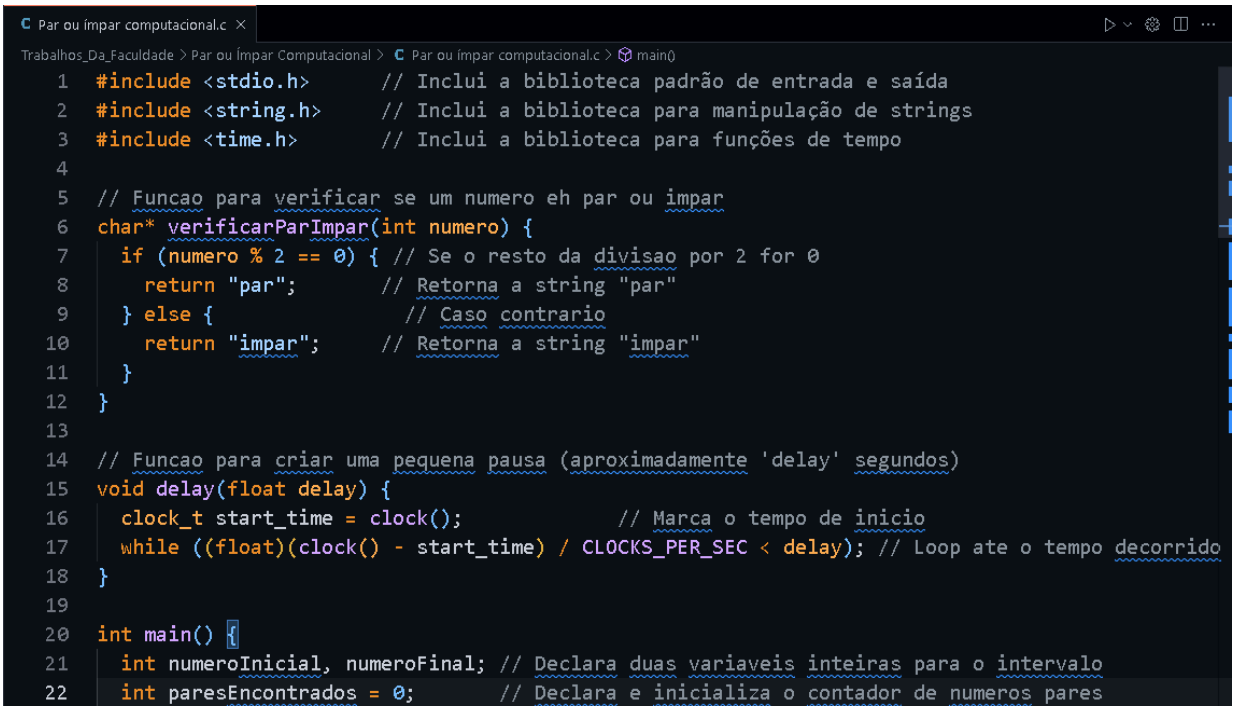
Para a determinação individual da paridade de cada número, foi concebida uma função específica, denominada **`verificarParImpar`**. Esta função recebe um número inteiro como parâmetro e aplica o operador módulo **(%)** para averiguar o resíduo da divisão desse número por 2. A ausência de resto (resultado igual a zero) indica que o número é classificado como par, enquanto a presença de um resto diferente de zero o define como ímpar. A função, então, comunica essa classificação através do retorno de uma string contendo o status ("**par**" ou "**impar**").

Durante a execução da estrutura de repetição, a função **`verificarParImpar`** é invocada para cada número pertencente ao intervalo. O status retornado é subsequentemente utilizado para construir e exibir uma mensagem informativa e clara ao usuário, apresentando o número corrente e sua respectiva classificação de paridade. Paralelamente a essa exibição, o programa mantém uma contagem precisa de todos os números pares identificados ao longo do processo iterativo.

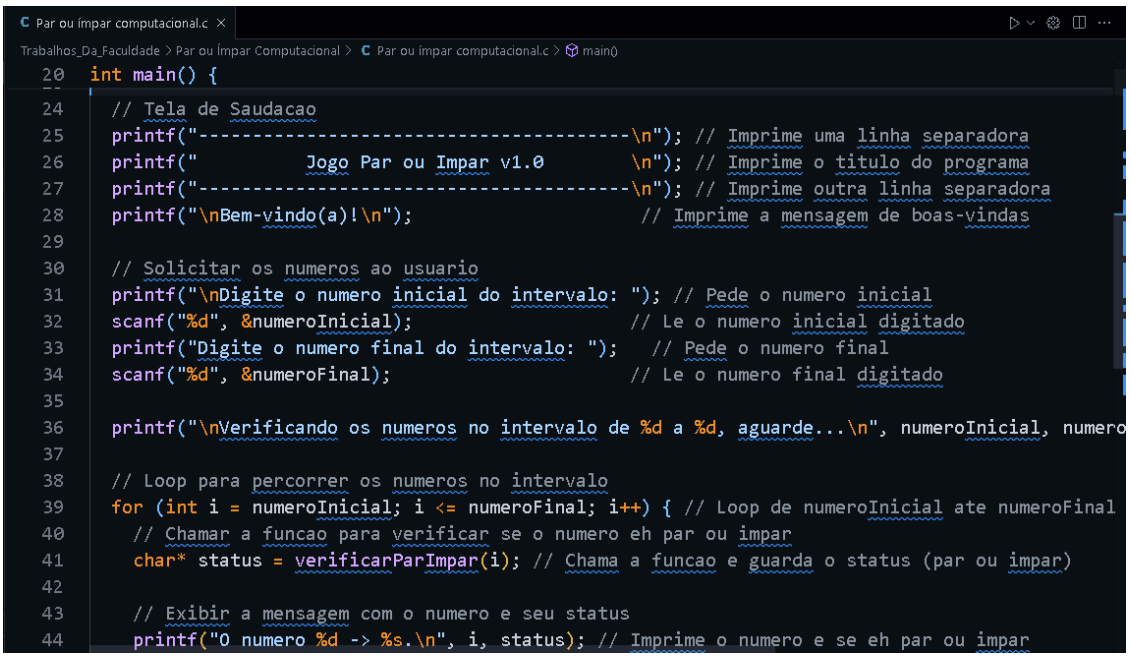
Visando aprimorar a experiência do usuário e tornar a execução mais perceptível, implementou-se uma função de atraso (**`delay`**). Esta função introduz uma breve pausa temporal após a apresentação de cada resultado, simulando um processo de análise progressiva e permitindo que o usuário acompanhe a verificação dos números de maneira mais espaçada e organizada.

Concluída a iteração e a análise de todos os números dentro do intervalo fornecido, o programa consolida as informações e exibe o total de números pares encontrados durante a execução. Por fim, uma mensagem de despedida cordial encerra a interação, buscando estabelecer uma sensação de conclusão amigável do processo.

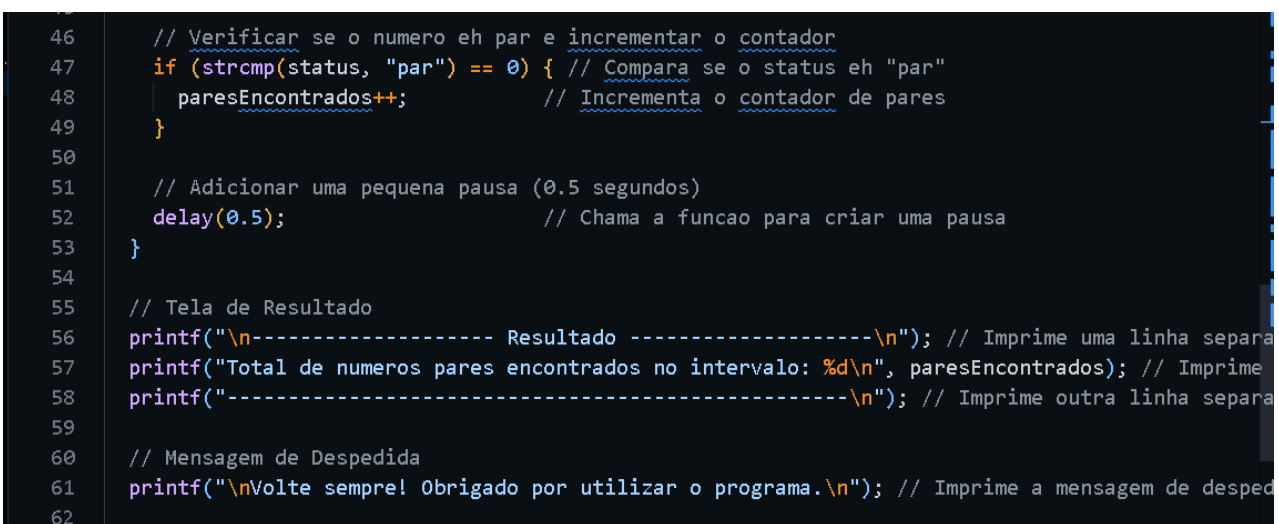
Anexos Visuais: Código Fonte



```c
#include <stdio.h>      // Inclui a biblioteca padrão de entrada e saída
#include <string.h>     // Inclui a biblioteca para manipulação de strings
#include <time.h>       // Inclui a biblioteca para funções de tempo

// Funcao para verificar se um numero eh par ou impar
char* verificarParImpar(int numero) {
  if (numero % 2 == 0) { // Se o resto da divisao por 2 for 0
    return "par";       // Retorna a string "par"
  } else {                // Caso contrario
    return "impar";     // Retorna a string "impar"
  }
}

// Funcao para criar uma pequena pausa (aproximadamente 'delay' segundos)
void delay(float delay) {
  clock_t start_time = clock();             // Marca o tempo de inicio
  while ((float)(clock() - start_time) / CLOCKS_PER_SEC < delay); // Loop ate o tempo decorrido
}

int main() {
  int numeroInicial, numeroFinal; // Declara duas variaveis inteiras para o intervalo
  int paresEncontrados = 0;       // Declara e inicializa o contador de numeros pares
```



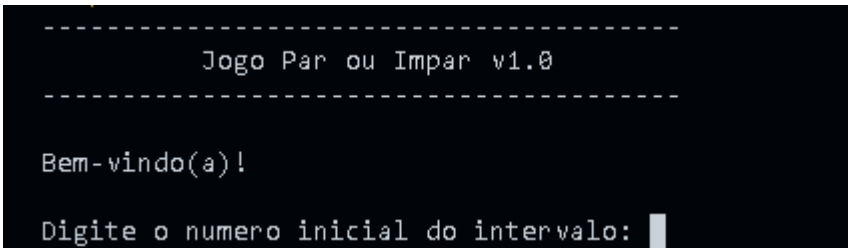
```c
int main() {
  // Tela de Saudacao
  printf("----------------------------------------\n"); // Imprime uma linha separadora
  printf("         Jogo Par ou Impar v1.0         \n"); // Imprime o titulo do programa
  printf("----------------------------------------\n"); // Imprime outra linha separadora
  printf("\nBem-vindo(a)!\n");                    // Imprime a mensagem de boas-vindas

  // Solicitar os numeros ao usuario
  printf("\nDigite o numero inicial do intervalo: "); // Pede o numero inicial
  scanf("%d", &numeroInicial);                   // Le o numero inicial digitado
  printf("Digite o numero final do intervalo: ");   // Pede o numero final
  scanf("%d", &numeroFinal);                     // Le o numero final digitado

  printf("\nVerificando os numeros no intervalo de %d a %d, aguarde...\n", numeroInicial, numero

  // Loop para percorrer os numeros no intervalo
  for (int i = numeroInicial; i <= numeroFinal; i++) { // Loop de numeroInicial ate numeroFinal
    // Chamar a funcao para verificar se o numero eh par ou impar
    char* status = verificarParImpar(i); // Chama a funcao e guarda o status (par ou impar)

    // Exibir a mensagem com o numero e seu status
    printf("O numero %d -> %s.\n", i, status); // Imprime o numero e se eh par ou impar
```



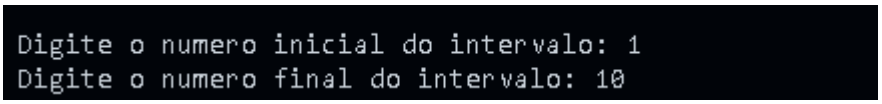
```c
    // Verificar se o numero eh par e incrementar o contador
    if (strcmp(status, "par") == 0) { // Compara se o status eh "par"
      paresEncontrados++;           // Incrementa o contador de pares
    }

    // Adicionar uma pequena pausa (0.5 segundos)
    delay(0.5);                     // Chama a funcao para criar uma pausa
  }

  // Tela de Resultado
  printf("\n-------------------- Resultado --------------------\n"); // Imprime uma linha separa
  printf("Total de numeros pares encontrados no intervalo: %d\n", paresEncontrados); // Imprime
  printf("-------------------------------------------------\n"); // Imprime outra linha separa

  // Mensagem de Despedida
  printf("\nVolte sempre! Obrigado por utilizar o programa.\n"); // Imprime a mensagem de desped
```

Lógica principal:

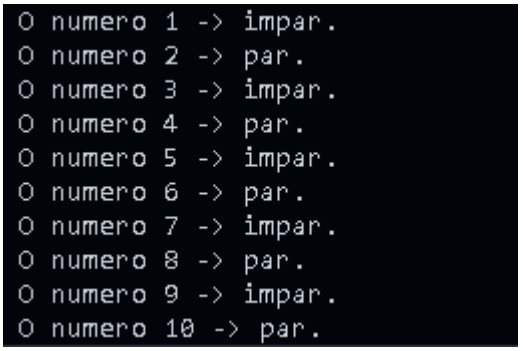
1 - Entrada de dados: O pede ao usuario para digitar o número desejado, uma tela de boas vindas foi adicionada, para a interface ficar mais amigável.



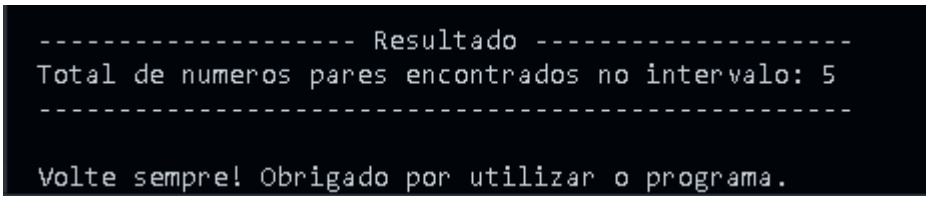
2 - Programa em ação : Exemplo de números menores, fiz o programa iniciar do 1 ao 10



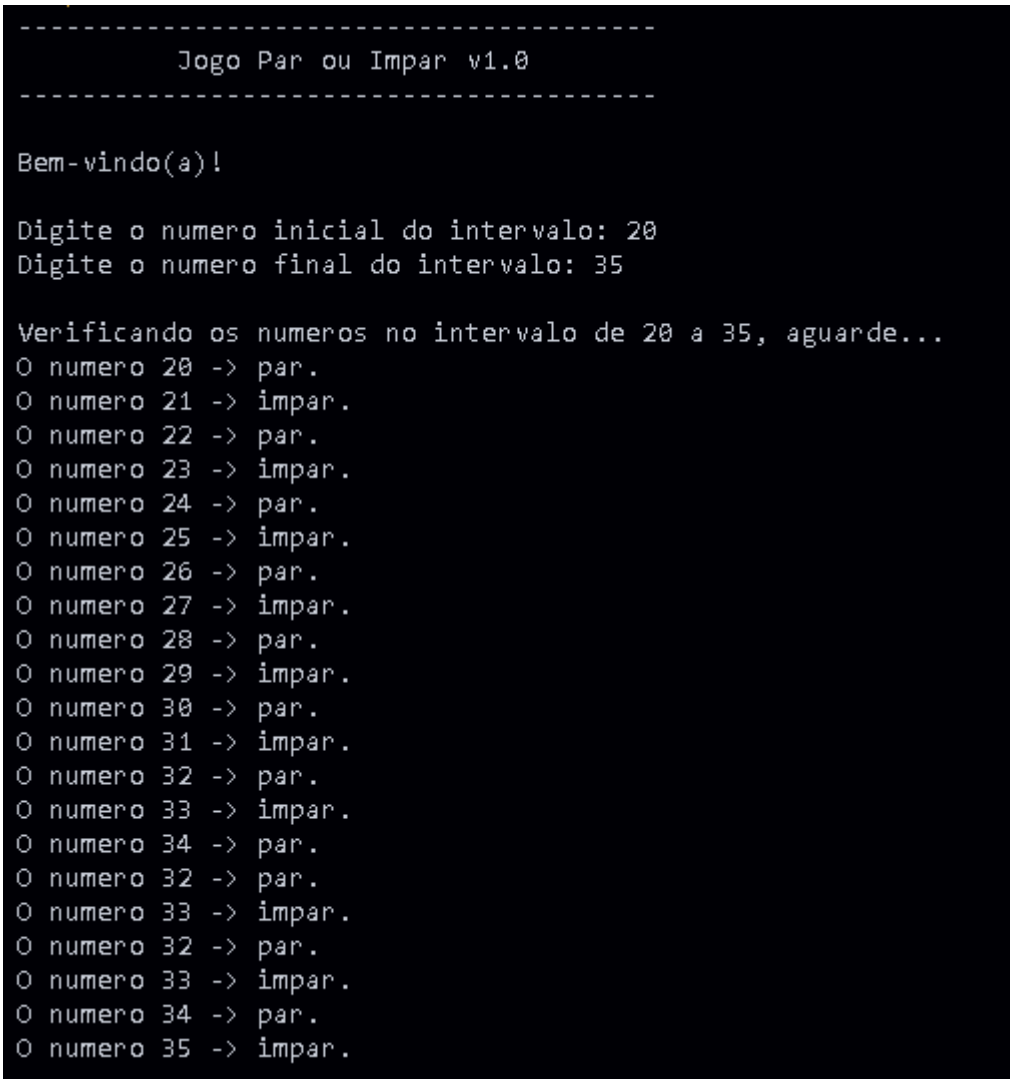
3 - Verificação dos números pares e ímpares:



4 - Resultado:



5 - Exemplo com números maiores:



```
----------------------------------------
          Jogo Par ou Impar v1.0
----------------------------------------

Bem-vindo(a)!

Digite o numero inicial do intervalo: 20
Digite o numero final do intervalo: 35

Verificando os numeros no intervalo de 20 a 35, aguarde...
O numero 20 -> par.
O numero 21 -> impar.
O numero 22 -> par.
O numero 23 -> impar.
O numero 24 -> par.
O numero 25 -> impar.
O numero 26 -> par.
O numero 27 -> impar.
O numero 28 -> par.
O numero 29 -> impar.
O numero 30 -> par.
O numero 31 -> impar.
O numero 32 -> par.
O numero 33 -> impar.
O numero 34 -> par.
O numero 32 -> par.
O numero 33 -> impar.
O numero 32 -> par.
O numero 33 -> impar.
O numero 34 -> par.
O numero 35 -> impar.
```

CONCLUSÃO

O presente trabalho demonstrou a aplicação eficaz da estrutura de repetição **`for`** em conjunto com a criação e utilização de uma função específica na linguagem de programação C para solucionar o problema da verificação de paridade em um intervalo numérico definido pelo usuário. Ao longo do desenvolvimento, observou-se a importância da interação clara com o usuário na obtenção dos dados de entrada, bem como a capacidade do loop **`for`** de iterar de maneira sistemática sobre um conjunto de valores, aplicando a lógica de verificação a cada elemento.

A função **`verificarParImpar`** desempenhou um papel crucial na modularização do código, encapsulando a lógica de determinação da paridade e promovendo a reutilização e a clareza do programa principal. A utilização do operador módulo (**`%`**) revelou-se uma ferramenta fundamental para a identificação precisa de números pares e ímpares. A inclusão da função **`delay`**, embora de caráter estético, contribuiu para uma experiência de usuário mais agradável, simulando um processamento gradual dos dados.

Os resultados obtidos durante a execução do programa demonstraram a sua capacidade de analisar corretamente a paridade de cada número dentro do intervalo especificado, além de contabilizar com precisão a quantidade total de números pares encontrados. A estrutura do código, aliada aos comentários detalhados, visa facilitar a compreensão do seu funcionamento e a demonstração do conhecimento adquirido sobre os conceitos abordados na disciplina. A implementação deste programa e os resultados obtidos estão alinhados com os princípios e exemplos discutidos no conteúdo da apostila da disciplina, em particular no que se refere à utilização eficiente da estrutura de repetição **`for`** para a iteração sobre sequências de dados e à criação de funções para a modularização do código.

Para o futuro desenvolvimento deste código, diversas melhorias e extensões poderiam ser consideradas...