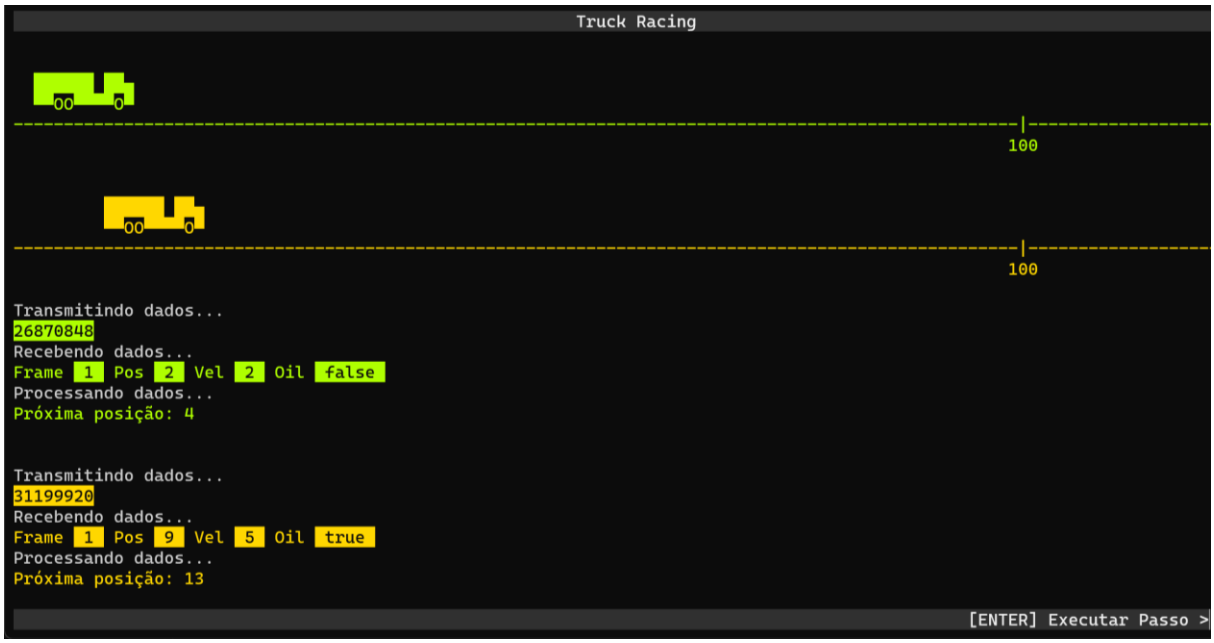
UNIVERSIDADE FEDERAL RURAL DO SEMIÁRIDO
CURSO DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMAÇÃO DE COMPUTADORES

TRABALHO PRÁTICO 1

No curso de Programação de Computadores, Joãozinho aprendeu que é possível desenhar caracteres gráficos usando a tabela ASCII e que estes caracteres podem ser coloridos através de códigos de *escape* definidos pela *American National Standards Institute* (ANSI). Ao ver que era possível desenhar um caminhão com estes caracteres, Joãozinho teve a ideia de criar um jogo de corrida entre caminhões e desenhou a seguinte tela:



Joãozinho também é um entusiasta de jogos multiplayer e, embora ele ainda não saiba como criar esse tipo de jogo, decidiu usar o conhecimento adquirido sobre operadores bit-a-bit para simular que seu jogo está enviando e recebendo dados pela rede, como se os dois caminhões fossem dois jogadores em máquinas diferentes.

Ajude Joãozinho a construir esse jogo, de forma que ele obedeça às seguintes regras:

- Os caminhões devem iniciar à esquerda da tela e se deslocar em passos.
- Cada passo deve deslocar o caminhão entre 0 e 10 caracteres para a direita.
- O pressionar de uma tecla deve executar um único passo de cada caminhão.
- Vence o caminhão que cruzar completamente a marca de 100 caracteres primeiro.

A ideia do jogo é simular a corrida. Cada caminhão terá uma posição e uma velocidade. A pista terá um estado que indicará se aquele trecho está com óleo na pista ou não. Além disso, deve-se manter controle sobre o número de passos que foram executados no jogo.

A cada passo, as seguintes tarefas devem ser realizadas:

1. Limpar a tela
2. Desenhar os caminhões em suas posições atuais
3. Atualizar a velocidade de cada caminhão usando um valor pseudoaleatório.

4. Atualizar o estado de cada pista usando um valor pseudoaleatório.
5. Codificar e transmitir os dados de cada caminhão/pista pela rede
6. Receber e decodificar os dados
7. Processar os dados
8. Encontrar e exibir a próxima posição de cada caminhão

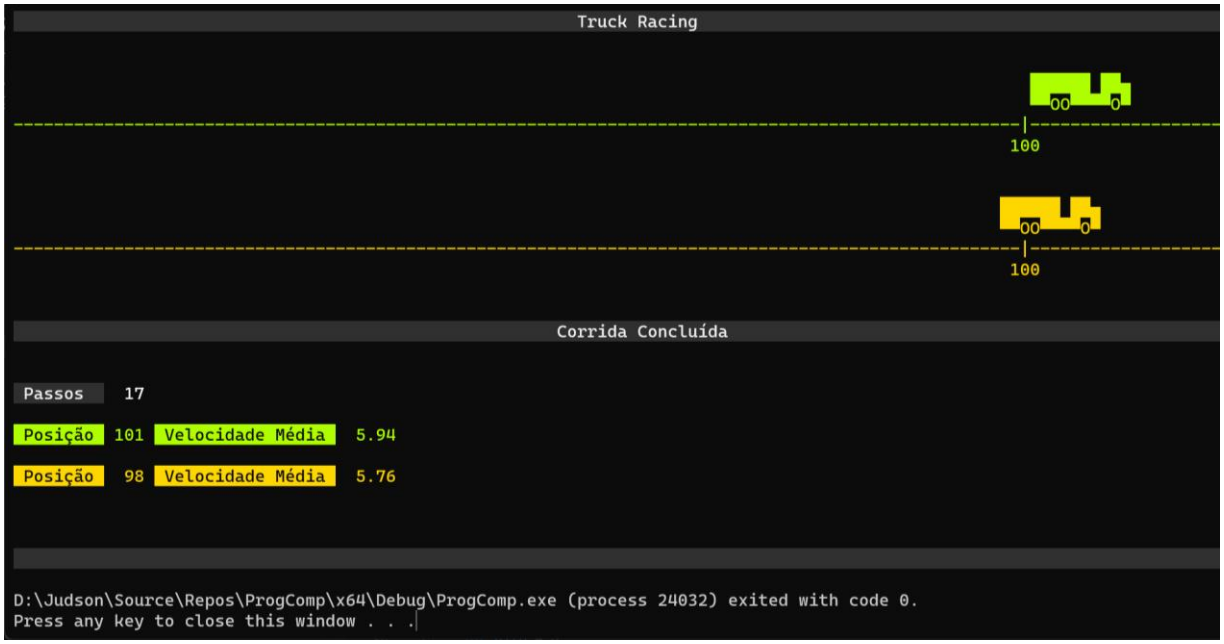
Antes de iniciar a sequência de passos, inicialize as variáveis necessárias realizar a simulação.

Ao final, exiba o número de passos da simulação juntamente com:

- A posição final de cada caminhão
- A velocidade média de cada caminhão
- O desenho dos caminhões em suas posições finais.

Utilize o trecho de código abaixo como um ponto de partida.

```
// --------------------
// Inicializa dados:
// - Posição
// - Velocidade
// - Cores
// - Estado da pista
// --------------------

int passo = 0;
do
{
    // ------------------
    // Limpa tela
    // Desenha caminhões
    // ------------------
    ...

    // ------------------------
    // Atualiza velocidade
    // Atualiza estado da pista
    // ------------------------
    ...

    // ---------------------------------
    // Transmite, recebe e processa dados
    // Calcula Próxima posição
    // ---------------------------------
    ...

    // -------------------------------
    // Aguarda pressionamento de tecla
    // -------------------------------

    passo = passo + 1;

}
while (posA <= 100 && posB <= 100);

// --------------------
// Limpa tela
// Desenha caminhões
// Exibe resultados
// --------------------
```



A simulação deve ser construída pela implementação das seguintes bibliotecas e funções.

Biblioteca Cores

- **Ajustar Cor**: função deve receber dois inteiros indicando a cor do texto e a cor do fundo. A função não deve retornar valores. Ela deve alterar a cor do terminal para a cor indicada.

- **Resetar Cor**: função que não recebe nem retorna valor. Ela deve modificar a cor do terminal para a sua cor padrão.

- Além das funções acima, a biblioteca Cores deve definir nomes para todas as cores que forem utilizadas no código. Dentro do arquivo .h utilize um #define para cada cor. As cores devem ser números de 0 a 255.

Biblioteca Pacote

- **Empacotar**: a função recebe cinco inteiros sem sinal, correspondentes ao número do passo, cor do caminhão, posição atual, velocidade e estado da pista. Ela deve retornar um inteiro de 32 bits, sem sinal, contendo as informações empacotadas com a organização do exemplo abaixo:

32 bits					
8 bits	8 bits	7 bits	4 bits	1 bit	4 bits
passo	cor	pos	vel	óleo	*padding*
1	220	9	5	1	0
31199920					

- **Passo**: a função recebe um inteiro de 32 bits sem sinal. Ela retorna o valor contido nos 8 primeiros bits da cadeia, como um valor inteiro sem sinal.

- **Cor**: a função recebe um inteiro de 32 bits sem sinal. Ela retorna o valor contido nos 8 bits subsequentes da cadeia, como um valor inteiro sem sinal.

- **Posição**: a função recebe um inteiro de 32 bits sem sinal. Ela retorna o valor contido nos 7 bits subsequentes da cadeia, como um valor inteiro sem sinal.

- **Velocidade**: a função recebe um inteiro de 32 bits sem sinal. Ela retorna o valor contido nos 4 bits subsequentes da cadeia, como um valor inteiro sem sinal.

- **Pista**: a função recebe um inteiro de 32 bits sem sinal. Ela retorna o valor contido no 1 bit subsequente da cadeia, como um valor inteiro sem sinal.

Biblioteca Rede

- **Transmitir**: a função recebe cinco inteiros, sem sinal, correspondentes ao número do passo, cor do caminhão, posição atual, velocidade e estado da pista. Ela não retorna valor. A função deve fazer uso da biblioteca Pacote e chamar uma função para empacotar os números. O valor de 32 bits resultante deve ser atribuído a uma variável global que representa a rede. Atribuir para esta variável significa transmitir na rede. A rede só aceita pacotes com 32 bits de largura.

- **Receber**: a função não recebe argumentos. Ela retorna um inteiro de 32 bits sem sinal. Ela deve retornar o valor contido na rede. Esse valor deve ser lido da variável global que representa a rede. Antes de retornar, a função deve decompor e exibir os dados (passo, posição, velocidade e estado da pista) usando as funções da biblioteca Pacote.

- **Processar**: a função recebe e retorna um inteiro de 32 bits sem sinal. O valor recebido corresponde a informação que veio da rede e deve ser processada. A função deve decompor os dados recebidos (posição, velocidade e estado da pista), calcular e retornar a próxima posição do caminhão com base na fórmula:

  nova_posição = posição_atual + velocidade – óleo;

  O valor de óleo deve ser 0 ou 1, a depender do estado da pista, com ou sem óleo. Veja que o caminhão pode ficar parado se a velocidade for igual a 1 e a pista estiver com óleo naquele trecho. O que o óleo faz é reduzir a velocidade em uma unidade. Os valores de velocidade devem ficar entre 1 e 10 para que essa conta não possa ficar negativa.

Programa Principal

O programa deve implementar o trecho de código apresentado anteriormente e fazer uso de duas funções, criadas diretamente no arquivo principal:

- **Desenhar**: a função recebe dois inteiros correspondentes a posição e cor do caminhão. Ela não retorna valor. A função deve fazer uso das funções da biblioteca Cores para ajustar a cor, desenhar o caminhão e retornar para a cor padrão após o desenho. Ela deve fazer uso da função Espaços para regular a quantidade de espaços em branco antes de cada caminhão.

- **Espaços**: a função deve ter a seguinte implementação, já fornecida:

```
void WhiteSpace(int qtd) { while (qtd-- > 0) cout << ' '; }
```

INSTRUÇÕES

1) As bibliotecas devem ser implementadas usando um arquivo .h com os protótipos das funções e um arquivo .cpp com a definição das funções. Se uma biblioteca precisar fazer uso de outras bibliotecas, seja ela uma biblioteca padrão ou uma criada por você, faça a inclusão da biblioteca no arquivo .cpp e não no arquivo .h.

2) Apenas as funções Transmitir e Receber devem acessar a variável global que representa a rede. Nenhuma outra variável global deve ser criada ou usada no programa. Crie a variável "rede" no mesmo .cpp onde estão as funções que precisam acessá-la.

3) Os nomes das bibliotecas, funções e variáveis descritas no trabalho são apenas sugestões. Sinta-se à vontade para usar os nomes que achar mais adequado. Eles podem, inclusive, ser em inglês.

4) Utilize as funções `srand` e `rand` para gerar números pseudoaleatórios. Faça uma única chamada a função `srand(unsigned(time(NULL))` no início do programa para ajustar o valor da semente usada pela função `rand`.

5) Utilize a função `system` com o argumento apropriado para limpar a tela e para fazer uma pausa no programa antes de cada novo passo. Pesquise quais seriam estes argumentos.

6) O design do programa não precisa ser igual ao apresentado nos exemplos, mas ele precisa conter textos e caracteres gráficos coloridos e alinhados. Utilize as funções apresentadas no exercício 4 da parte de fixação do Lab02 para o alinhamento de textos.

7) Não utilize nada que ainda não foi mostrado durante as aulas e exercícios da disciplina.

ENTREGA DO TRABALHO

Grupos: Trabalho individual

Data da entrega: 05/02/2024 (até a meia noite)

Valor do Trabalho: 3,0 pontos (na 1a Unidade)

Forma de entrega: enviar apenas os arquivos fonte (.cpp) e os arquivos de inclusão (.h) compactados no formato **zip** através da tarefa correspondente no SIGAA.

O não cumprimento das orientações resultará em **penalidades:**

- Programa não executa no Visual Studio 2022 (3,0 pontos)
- Programa contém partes de outros trabalhos (3,0 pontos)
- Programa utiliza recursos não vistos na disciplina (3,0 pontos)
- Atraso na entrega (1,5 pontos por dia de atraso)
- Arquivo compactado em outro formato que não zip (0,5 ponto)
- Envio de outros arquivos que não sejam os .cpp e .h (0,5 ponto)
- Programa sem comentários e/ou desorganizado (0,5 ponto)