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 manipular a cadeia de bits dentro de um número inteiro usando operadores bit-a-bit. Ele também viu que a tabela ASCII possui alguns caracteres gráficos que podem ser coloridos através de códigos de *escape* definidos pela *American National Standards Institute* (ANSI). Na Internet, Joãozinho aprendeu que a tela do computador é formada por pequenos pontos coloridos chamados de pixels e percebeu que um conjunto de bits poderia muito bem representar os pixels que estão ligados e desligados na tela do computador. Para explorar todas essas possibilidades Joãozinho resolveu criar um programa que exiba bits de forma visual em uma tela. Ajude Joãozinho nessa jornada, fazendo o que se pede.

INICIALIZAÇÃO

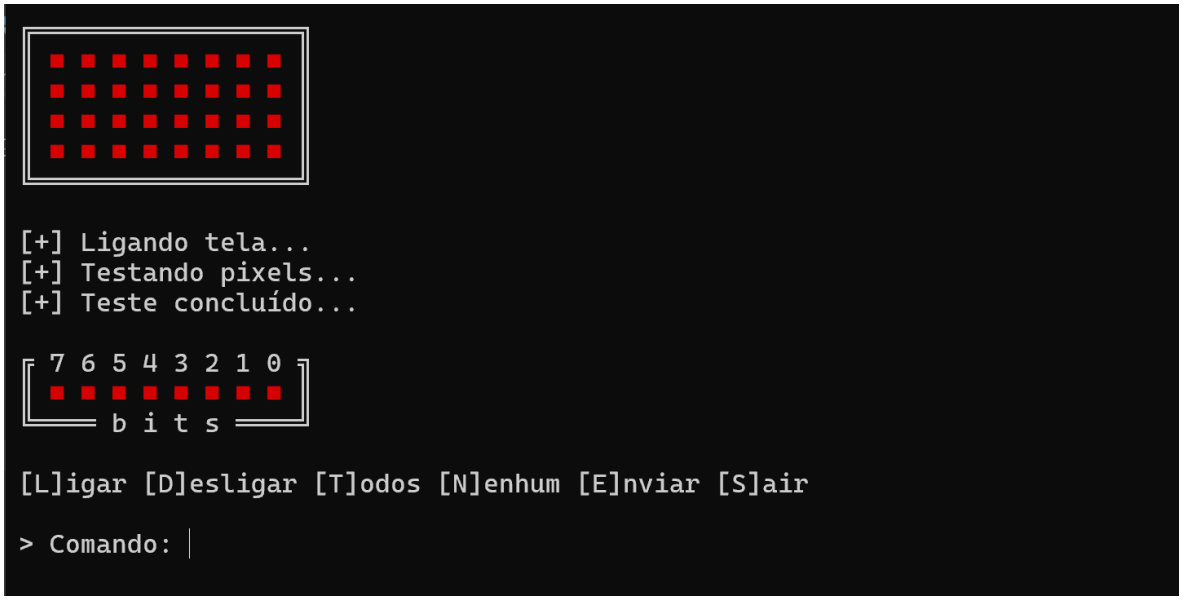
O programa deve iniciar exibindo uma tela com todos os seus 32 pixels ligados. A tela deve ser formada por 4 linhas, com 8 pixels em cada linha. Utilize uma cor para representar os pixels ligados e uma cor diferente para representar os pixels desligados. Desenhe uma moldura em volta dos pixels para representar a tela.

Depois de 1.5 segundos, a tela deve passar para o modo de teste automático. Nesse modo ela deve trocar sua imagem 10 vezes, com um intervalo de 1 segundo entre cada troca. Cada imagem deve ser representada por um valor inteiro de 32 bits gerado aleatoriamente.

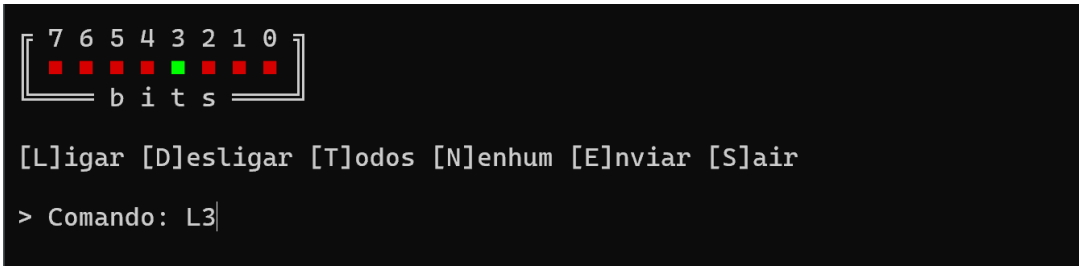
Uma vez concluído os testes, o programa deve desligar todos os pixels da tela e exibir uma interface para que o usuário possa manipular uma linha (8 bits) da tela por vez através de comandos lidos pelo teclado.

FUNCIONAMENTO

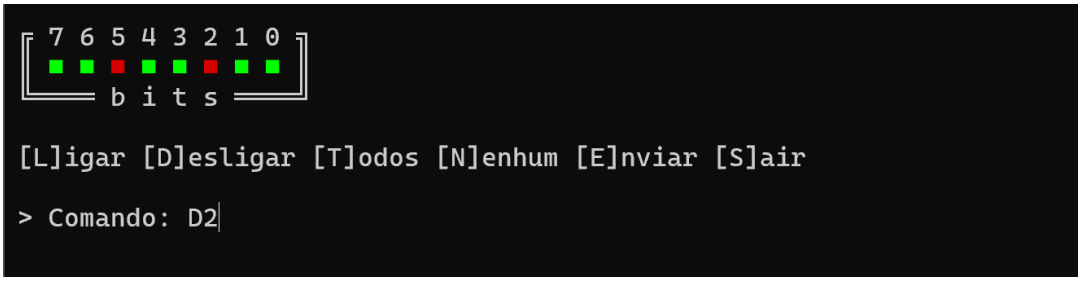
Os **comandos** devem ser acionados por uma letra (maiúscula) seguida opcionalmente por um número e sempre finalizados pela tecla ENTER. A janela do prompt de comandos deve ser limpa e redesenhada toda vez que um comando for executado. Os comandos devem permitir ligar um pixel, desligar um pixel, ligar todos os pixels da linha, desligar todos os pixels da linha, enviar uma linha para a tela e sair do programa.



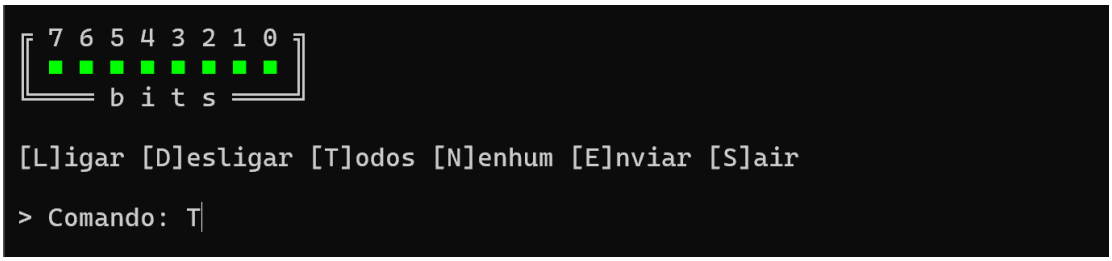
- **Ligar**: o comando **L** deve ser seguido por um número de 0 a 7 e deve ligar o bit correspondente na fita de bits exibida na tela.



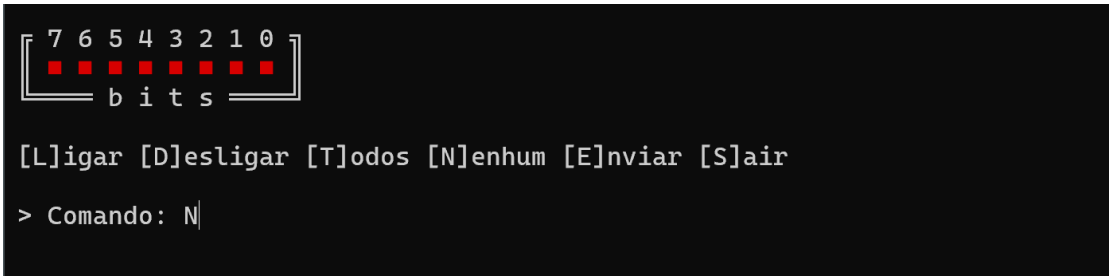
- **Desligar**: o comando **D** deve ser seguido por um número de 0 a 7 e deve desligar o bit correspondente na fita de bits exibida na tela.



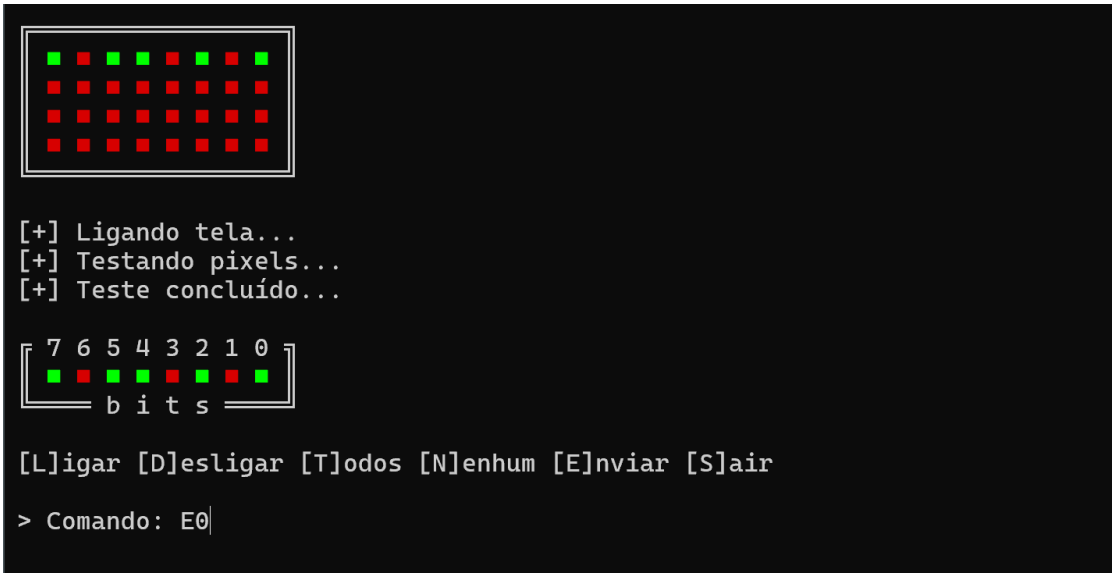
- **Todos**: o comando **T** deve ligar todos os bits da fita.



- **Nenhum**: o comando **N** deve desligar todos os bits da fita.



- **Enviar**: o comando **E** deve ser seguido por um número de 0 a 3 e deve enviar os dados da fita de bits para a linha correspondente da tela.



- **Sair**: o comando **S** deve encerrar o programa, levando-o para a etapa de finalização. Na prática, ele vai fazer o programa sair do loop em que ele estava preso. Esse loop permite que o programa se mantenha aceitando novos comandos até que o usuário execute o comando de saída.

FINALIZAÇÃO

Na finalização, o programa deve simular um ataque Hacker. Como se Joãozinho não fosse o autor do programa, mas sim um invasor que consegue acessar a cadeia de bits que forma a imagem na tela.

O inteiro de 32 bits que guarda a imagem da tela deve ser exibido em formato binário. Para isso pesquise sobre a biblioteca <bitset> do C++ e a utilize junto com cout para mostrar o código binário do número inteiro.

Utilize funções “Bits Altos” e “Bits Baixos” para separar o número de 32 bits em duas variáveis de 16 bits. Crie uma função “Hackear” que exiba as informações de metade da tela, ou seja, que receba um inteiro de 16 bits em vez de um de 32 bits e mostre duas linhas de pixels em vez de quatro. A função deve ser chamada duas vezes recebendo em cada chamada uma das variáveis de 16 bits obtidas anteriormente.

Observe pelo exemplo que a função não mostrou os pixels tradicionais na tela, ela desenhou letras para o nome do Hacker em algumas posições selecionadas. Fique à vontade para usar a criatividade, mas mantenha as cores dos pixels originais. Finalmente, não esqueça de mostrar uma mensagem do Hacker e emita um som usando o caractere especial da tabela ASCII chamado de BEL.

IMPLEMENTAÇÃO

Abaixo é exibida a estrutura básica da função principal. Ela usa um laço para repetir código uma quantidade fixa de vezes (for) e outro laço que executa uma quantidade indefinida de vezes (while). A implementação da etapa de teste da tela deve ser realizada no primeiro laço. A captura e tratamento dos comandos, bem como o desenho da tela e fita de bits deve ser realizada dentro do segundo laço. Como esses assuntos ainda não foram vistos na disciplina, os laços já estão sendo fornecidos prontos para uso.

```
// inclusão de bibliotecas

// criando funções com macros
#define Print(X) cout << X
#define Read(X) cin >> X

int main()
{
    // mostra a tela ligada

    for (int i = 0; i < 10; i++)
    {
        // limpa a janela
        // exibe tela com pixels aleatórios
        // dorme por 1 segundo
    }

    // desliga pixels da tela
    // mostra interface de comandos

    int num;
    char op;
    Read(op);

    // repete até ser digitado 'S'
    while (op != 'S')
    {
        // testa o comando escolhido e executa a tarefa
        // limpa a janela
        // redesenha a tela
        // reexibe os textos
        // mostra a interface de comandos

        Read(op);
    }

    // finaliza o programa
    // hackeando a tela
}
```

FUNÇÕES E BIBLIOTECAS

O programa deve implementar as seguintes bibliotecas e funções.

Biblioteca Bits

- **Bits Altos**: recebe um inteiro de 32 bits sem sinal e retorna os 16 bits mais a esquerda (mais altos) em um inteiro de 16 bits sem sinal.

- **Bits Baixos**: recebe um inteiro de 32 bits sem sinal e retorna os 16 bits mais a direita (mais baixos) em um inteiro de 16 bits sem sinal.

- **Testar Bit**: recebe um inteiro de 8 bits sem sinal representando o estado atual da fita de bits, um inteiro de 8 bits sem sinal representando o número do bit a ser testado e retorna um booleano indicando se o bit está ou não ligado.

- **Ligar Bit**: recebe um inteiro de 8 bits sem sinal representando o estado atual da fita de bits, um inteiro de 8 bits sem sinal representando o número do bit a ser ligado e retorna um inteiro de 8 bits sem sinal representando o novo estado da fita com o bit já ligado.

- **Desligar Bit**: recebe um inteiro de 8 bits sem sinal representando o estado atual da fita de bits, um inteiro de 8 bits sem sinal representando o número do bit a ser desligado e retorna um inteiro de 8 bits sem sinal representando o novo estado da fita com o bit já desligado.

Biblioteca Tela

- **Linha**: recebe um inteiro de 8 bits sem sinal representando uma linha da tela e desenha a linha, usando caracteres gráficos com uma cor para os bits ligados e com outra cor para os bits desligados.

- **Fita**: recebe um inteiro de 8 bits sem sinal representando a fita de bits e desenha a fita na tela com todos os seus adornos. A função Fita deve fazer uso da função Linha.

- **Tela**: recebe um inteiro de 32 bits sem sinal representando o estado atual dos pixels da tela e desenha a tela com todos os seus adornos. A função Tela deve quebrar o inteiro de 32 bits em 4 inteiros de 8 bits e usar a função Linha para desenhar cada linha da tela.

- **Atualizar**: a função deve receber 3 argumentos, um inteiro de 32bits sem sinal representando o estado atual dos pixels na tela, um inteiro de 8 bits sem sinal representando o estado atual da fita de bits e um inteiro representando o número da linha da tela (valor entre 0 e 3) a ser atualizada com os bits da fita. Ela deve retornar um inteiro de 32 bits sem sinal representando o novo estado da tela, que deve ter sido atualizado com os bits da fita sobrescrevendo os bits da linha indicada.

- **LinhaHacker**: recebe um inteiro de 8 bits sem sinal e exibe uma linha de pixels personalizada pelo Hacker.

- **Hackear**: recebe um inteiro de 16 bits sem sinal, separa os dados em dois inteiros de 8 bits sem sinal e chama a função LinhaHacker para cada uma das partes, exibindo assim duas linhas de pixels personalizadas pelo Hacker.

Biblioteca Random

- **Srand32**: utiliza as funções `time` e `srand` das bibliotecas padrão do C++ para gerar uma semente para o gerador de números pseudoaleatórios `rand`.

- **Rand32**: utiliza a função `rand` da biblioteca padrão do C++ em conjunto com operações matemáticas e operadores bit-a-bit para construir um número aleatório com 32 bits.

  A função `rand` retorna um valor na faixa de 0 a 32767. Isso quer dizer que ela retorna um valor que utiliza apenas 15 bits de um inteiro. Se usarmos a função `rand` duas vezes, podemos concatenar os resultados para obter um valor com 30 bits. Sorteando mais dois valores na faixa de 0 a 1, podemos completar um inteiro com 32 bits aleatórios:

32 bits			
1 bit	15 bits	1 bit	15 bits
Valor 0 ou 1	Valor de 0 a 32767	Valor 0 ou 1	Valor de 0 a 32767

OBSERVAÇÕES

1) As funções solicitadas devem ser usadas para executar as tarefas do programa. A não utilização das funções resultará em descontos de pontos equivalentes a não implementação das funções.

2) As bibliotecas devem ser implementadas usando um arquivo .h para os protótipos das funções e um arquivo .cpp para a definição das funções. Se uma biblioteca precisar fazer uso de outras bibliotecas, seja uma biblioteca padrão ou uma criada no programa, faça a inclusão da biblioteca no arquivo .cpp, nunca no arquivo .h.

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 se desejar praticar a língua.

4) Não utilize as funções `srand` e `rand` para gerar números pseudoaleatórios no programa principal. Faça uso apenas das funções Srand32 e Rand32 criadas na sua biblioteca Random. `srand` e `rand` devem ser usadas apenas na criação da biblioteca Random.

5) Para limpar a janela do programa utilize a função `system` com o argumento apropriado.

6) Pesquise e use uma função que permite ao programa dormir por um tempo determinado para fazer as pausas de tempo solicitadas no programa.

7) O design do programa não precisa ser igual ao apresentado nos exemplos, mas ele precisa conter caracteres gráficos coloridos e alinhados de forma a representar uma tela formada por pixels.

8) Não utilize nada que não foi mostrado ou usado durante as aulas ou exercícios da disciplina. Cuidado com códigos prontos obtidos de serviços como ChatGPT, Copilot, Gemini ou similares, eles tendem a usar recursos do C++ que não foram abordados no curso.

ENTREGA DO TRABALHO

Grupos: Trabalho individual

Data da entrega: 06/10/2025 (até a meia noite)

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

Forma de entrega:

Ativar a exibição de arquivos ocultos no explorador de arquivos, apagar a pasta oculta .vs da pasta da solução, apagar a pasta x64 da pasta da solução, apagar a pasta x64 da pasta do projeto, apagar arquivo com a extensão .suo, se ele existir em qualquer uma das pastas.

Após as exclusões, confira se os arquivos restantes, que estão na pasta da solução e na pasta do projeto, têm uma das extensões a seguir. Nenhum outro tipo de arquivo deve ser enviado.

- .h – arquivo de cabeçalho
- .cpp – arquivo de código fonte
- .slnx – solução do visual studio
- .vcxproj – projeto do visual studio
- .vcxproj.filters – filtros do projeto do visual studio
- .vcxproj.user – configurações do projeto do visual studio

Em seguida compactar todo o conteúdo da pasta da solução em um **arquivo .zip** (não usar .rar, .7zip ou outros formatos de compressão) e enviar através da tarefa correspondente no SIGAA.

Penalidades:

O não cumprimento das orientações acima resultarão em descontos na pontuação do trabalho:

- 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 ou pastas que não sejam os descritos acima (0,5 ponto)
- Programa sem comentários (0,5 ponto)
- Código sem indentação e/ou desorganizado (0,5 ponto)
- Nomes de variáveis e funções sem significado (0,5 ponto)