Este é o CS50

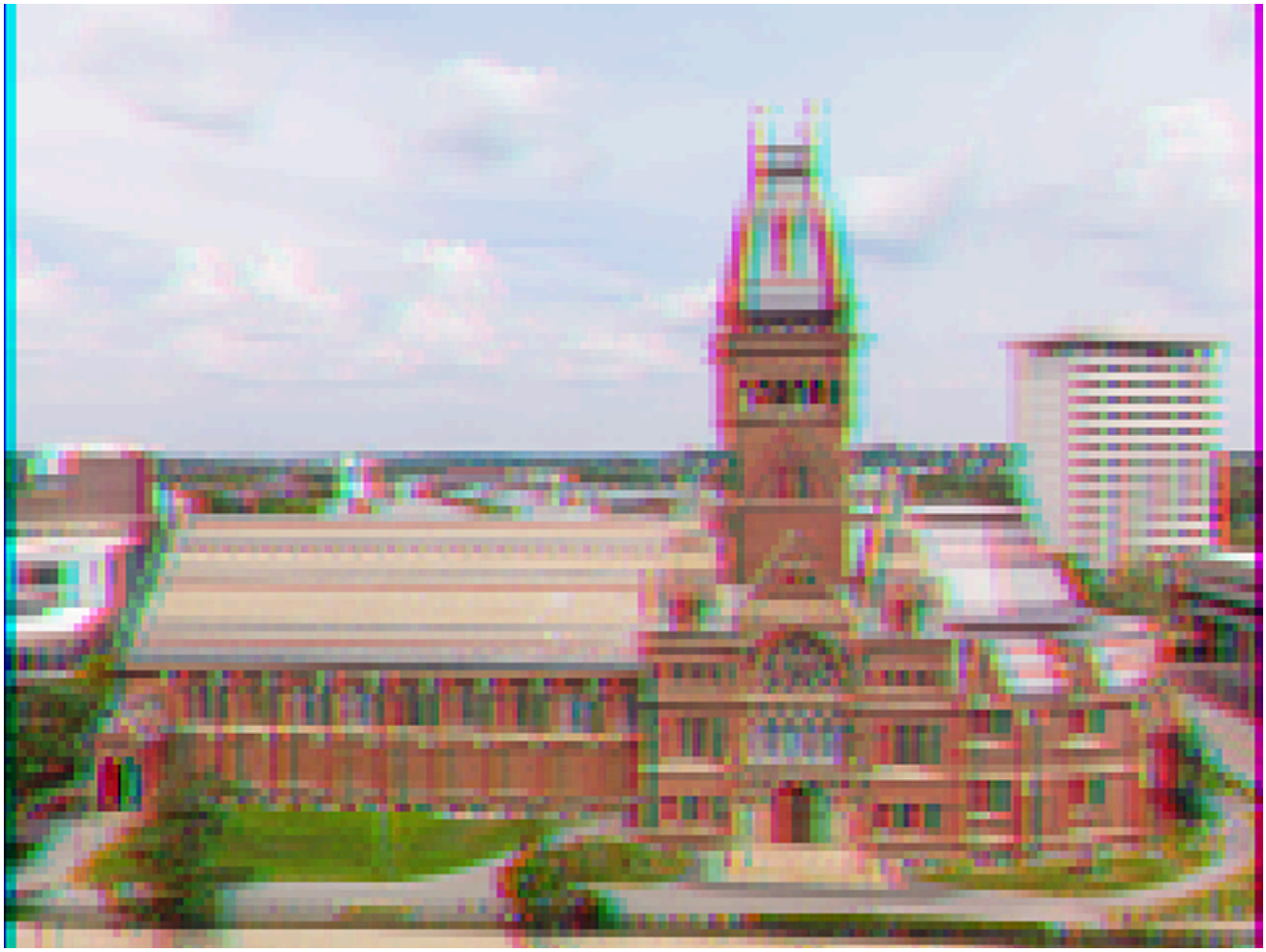
Introdução à Ciência da Computação (CS50)

OpenCourseWare

Doar (https://cs50.harvard.edu/donate)

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.com/dmalan) (https://github.com/dmalan)
(https://www.instagram.com/davidjmalan/) (https://www.linkedin.com/in/malan/)
(https://www.reddit.com/user/davidjmalan)
(https://www.threads.net/@davidjmalan) (https://twitter.com/davidjmalan)

Recuperar



Problema a resolver

Prevendo esse problema, passamos os últimos dias tirando fotos pelo campus, todas salvas em formato JPEG em um cartão de memória. Infelizmente, acabamos apagando todas! Felizmente, no mundo da informática, "apagado" geralmente significa menos "apagado" e mais "esquecido". Embora a câmera indique que o cartão está vazio, temos quase certeza de que não é bem assim. Aliás, estamos torcendo (ou melhor, esperando!) que você consiga escrever um programa para recuperar as fotos para nós!

Em um arquivo chamado `<nome_do_arquivo>` `recover.c` dentro de uma pasta chamada `<nome_da_pasta>` `recover`, escreva um programa para recuperar arquivos JPEG de um cartão de memória.

Código de Distribuição

Para este problema, você deverá estender a funcionalidade do código fornecido pela equipe do CS50.

▼ **Baixe o código de distribuição.**

Faça login em cs50.dev (https://cs50.dev/) , clique na janela do terminal e execute `cd` o comando. Você verá que o prompt do terminal será semelhante ao seguinte:

```
$
```

Em seguida, execute

```
wget https://cdn.cs50.net/2025/x/psets/4/recover.zip
```

para baixar um arquivo ZIP chamado `recover.zip` para o seu espaço de código.

Em seguida, execute

```
unzip recover.zip
```

para criar uma pasta chamada `recover`. Você não precisa mais do arquivo ZIP, então pode executar

```
rm recover.zip
```

e responda com “y” seguido de Enter quando solicitado para remover o arquivo ZIP que você baixou.

Agora digite

```
cd recover
```

Em seguida, pressione Enter para entrar (ou seja, abrir) esse diretório. Seu prompt agora deve ser semelhante ao abaixo.

```
recover/ $
```

Execute o comando `ls` sozinho e você deverá ver dois arquivos: `recover.c` e `card.raw`.

Fundo

Embora os JPEGs sejam mais complexos que os BMPs, eles possuem "assinaturas", padrões de bytes que os distinguem de outros formatos de arquivo. Especificamente, os três primeiros bytes de um JPEG são

```
0xff 0xd8 0xff
```

do primeiro ao terceiro byte, da esquerda para a direita. O quarto byte, por sua vez, é `0xe0`, `0xe1`, `0xe2`, `0xe3`, `0xe4`, `0xe5`, `0xe6`, `0xe7`, `0xe8`, `0xe9`, `0xea`, `0xeb`, `0xec`, `0xed`, `0xee` ou `0xef`. Em outras palavras, os quatro primeiros bits do quarto byte são `1110`.

É bem provável que, se você encontrar esse padrão de quatro bytes em mídias conhecidas por armazenar fotos (por exemplo, meu cartão de memória), eles demarquem o início de um arquivo JPEG. Para ser justo, você pode encontrar esses padrões em algum disco por puro acaso, então a recuperação de dados não é uma ciência exata.

Felizmente, as câmeras digitais tendem a armazenar as fotografias de forma contígua nos cartões de memória, de modo que cada foto é armazenada imediatamente após a anterior. Assim, o início de um JPEG geralmente marca o fim de outro. No entanto, as câmeras digitais frequentemente inicializam os cartões com um sistema de arquivos FAT, cujo "tamanho do bloco" é de 512 bytes (B). Isso significa que essas câmeras gravam nos cartões apenas em unidades de 512 B. Uma foto de 1 MB (ou seja, 1.048.576 B) ocupa, portanto, 1.048.576 ÷ 512 = 2.048 "blocos" em um cartão de memória. Mas o mesmo acontece com uma foto que seja, digamos, um byte menor (ou seja, 1.048.575 B)! O espaço desperdiçado no disco é chamado de "espaço livre". Investigadores forenses frequentemente examinam o espaço livre em busca de vestígios de dados suspeitos.

A implicação de todos esses detalhes é que você, o investigador, provavelmente pode escrever um programa que itere sobre uma cópia do meu cartão de memória, procurando por assinaturas de JPEGs. Cada vez que encontrar uma assinatura, você pode abrir um novo arquivo para escrita e começar a preenchê-lo com bytes do meu cartão de memória, fechando-o apenas quando encontrar outra assinatura. Além disso, em vez de ler os bytes do meu cartão de memória um por vez, você pode ler 512 deles por vez em um buffer para maior eficiência. Graças à FAT, você

pode confiar que as assinaturas dos JPEGs estarão "alinhadas a blocos". Ou seja, você só precisa procurar por essas assinaturas nos primeiros quatro bytes de um bloco.

É claro que os arquivos JPEG podem abranger blocos contíguos. Caso contrário, nenhum JPEG poderia ter mais de 512 bytes. Mas o último byte de um JPEG pode não estar exatamente no final de um bloco. Lembre-se da possibilidade de haver espaço livre. Mas não se preocupe. Como este cartão de memória era novo quando comecei a tirar fotos, é provável que ele tenha sido "zerado" (ou seja, preenchido com zeros) pelo fabricante, e nesse caso, qualquer espaço livre estará preenchido com zeros. Não tem problema se esses zeros finais aparecerem nos JPEGs que você recuperar; eles ainda poderão ser visualizados.

Bem, eu só tenho um cartão de memória, mas vocês são muitos! Então, criei uma "imagem forense" do cartão, armazenando seu conteúdo, byte a byte, em um arquivo chamado `card.raw`. Para que vocês não percam tempo iterando sobre milhões de zeros desnecessariamente, criei a imagem apenas dos primeiros megabytes do cartão de memória. Mas vocês devem encontrar, ao final, 50 arquivos JPEG.

Especificação

Implemente um programa `recover` que recupere arquivos JPEG de uma imagem forense.

- Implemente seu programa em um arquivo chamado `recover.c` em um diretório chamado `recover`.
- Seu programa deve aceitar exatamente um argumento de linha de comando, o nome de uma imagem forense da qual recuperar os arquivos JPEG.
- Se o seu programa não for executado com exatamente um argumento de linha de comando, ele deverá lembrar o usuário do uso correto e `main` retornar um erro `1`.
- Se a imagem forense não puder ser aberta para leitura, seu programa deverá informar o usuário e `main` retornar um erro `1`.
- Os arquivos que você gerar devem ser nomeados como `###.jpg`, onde `###` é um número decimal de três dígitos, começando com `000` para a primeira imagem e continuando em ordem crescente.
- Seu programa, caso utilize `malloc`, não deve apresentar vazamento de memória.

Dicas

Clique nos botões abaixo para ler algumas dicas!

▼ **Escreva um pseudocódigo antes de escrever mais código.**

Se não tiver certeza de como resolver o problema maior, divida-o em problemas menores que você provavelmente conseguirá resolver primeiro. Por exemplo, este problema, na verdade, se resume a apenas alguns problemas:

1. Aceita um único argumento de linha de comando: o nome de um cartão de memória.
2. Abra o cartão de memória.
3. Embora ainda haja dados para ler no cartão de memória
    1. Criar JPEGs a partir dos dados

Vamos escrever um pseudocódigo como comentários para lembrá-lo de fazer exatamente isso:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    // Accept a single command-line argument

    // Open the memory card

    // While there's still data left to read from the memory card

        // Create JPEGs from the data
}
```

▼ **Converta o pseudocódigo em código.**

Primeiramente, considere como aceitar um único argumento de linha de comando. Se o usuário fizer uso indevido do programa, você deve orientá-lo sobre o uso correto do programa.

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    // Accept a single command-line argument
    if (argc != 2)
    {
        printf("Usage: ./recover FILE\n");
        return 1;
    }

    // Open the memory card

    // While there's still data left to read from the memory card

        // Create JPEGs from the data
}
```

Agora que você verificou se o uso está correto, pode abrir o cartão de memória. Lembre-se de que você pode abri-lo `card.raw` programaticamente com `fopen` o comando abaixo.

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
```

```
    // Accept a single command-line argument
    if (argc != 2)
    {
        printf("Usage: ./recover FILE\n");
        return 1;
    }

    // Open the memory card
    FILE *card = fopen(argv[1], "r");

    // While there's still data left to read from the memory card

        // Create JPEGs from the data
}
```

É claro que você deve verificar se o arquivo foi aberto corretamente! Caso contrário, avise o usuário e encerre o programa: deixaremos essa parte por sua conta.

Em seguida, seu programa deve ler os dados do cartão que você abriu, até que não haja mais dados para ler. Ao longo do processo, seu programa deve recuperar cada um dos arquivos JPEG `card.raw`, armazenando cada um como um arquivo separado em seu diretório de trabalho atual.

Primeiro, considere como ler `card.raw` os dados do início ao fim. Lembre-se de que, para ler dados de um arquivo, você precisa armazená-los temporariamente em um "buffer". E lembre-se também de que `card.raw` os dados são armazenados em blocos de 512 bytes. Portanto, você provavelmente desejará criar um buffer de 512 bytes para armazenar os blocos de dados à medida que os lê sequencialmente. Uma maneira de fazer isso é usar o `uint8_t` tipo `int` da biblioteca `std::vector` `stdint.h`, que armazena exatamente 8 bits (1 byte). O tipo `int` é chamado `uint8_t` assim porque armazena um inteiro sem sinal, positivo e não negativo que requer 8 bits de espaço (ou seja, um byte).

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    // Accept a single command-line argument
    if (argc != 2)
    {
        printf("Usage: ./recover FILE\n");
        return 1;
    }

    // Open the memory card
    FILE *card = fopen(argv[1], "r");

    // Create a buffer for a block of data
    uint8_t buffer[512];

    // While there's still data left to read from the memory card


```

```
        // Create JPEGs from the data
}
```

Provavelmente não é a melhor ideia usar 512 como um "número mágico" aqui. É bem provável que você consiga aprimorar ainda mais esse projeto!

Agora, considere como ler dados do cartão de memória. De acordo com a página do manual (https://man.cs50.io/3/fread) , `fread` a função `read_bytes` retorna o número de bytes lidos, caso em que deve retornar `NULL` `512` ou `0` `NULL`, visto que `card.raw` o cartão contém vários blocos de 512 bytes. Para ler todos os blocos do cartão `card.raw`, após abri-lo com `read_bytes` `fopen`, basta usar um loop como este.

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    // Accept a single command-line argument
    if (argc != 2)
    {
        printf("Usage: ./recover FILE\n");
        return 1;
    }

    // Open the memory card
    FILE *card = fopen(argv[1], "r");

    // Create a buffer for a block of data
    uint8_t buffer[512];

    // While there's still data left to read from the memory card
    while (fread(buffer, 1, 512, card) == 512)
    {
        // Create JPEGs from the data

    }
}
```

Dessa forma, assim que `fread` retornar `0` (o que é efetivamente `false`), seu loop terminará.

Por fim, cabe a você determinar como criar JPEGs programaticamente à medida que continua a ler os dados . Para isso, o passo a passo `card.raw` abaixo pode ser útil.

`###.jpg` Lembre-se de que seu programa deve numerar os arquivos que gera , nomeando cada um com `###` um número decimal de três dígitos `000`, a partir de 0. Observe `sprintf` (https://man.cs50.io/3/sprintf)que a função `array` `sprintf` armazena uma string formatada em um local da memória. Dado o `###.jpg` formato prescrito para o nome de um arquivo JPEG, quantos bytes você deve alocar para essa string? (Não se esqueça do caractere NUL!)

Para verificar se os arquivos JPEG gerados pelo seu programa estão corretos, basta clicar duas vezes e conferir! Se cada foto aparecer intacta, sua operação provavelmente foi um sucesso!

E, claro, lembre-se de salvar `fclose` todos os arquivos que você abriu com o comando `fopen`!

▼ **Mantenha seu diretório de trabalho organizado.**

É provável que os arquivos JPEG gerados pela primeira versão do seu código estejam incorretos. (Se você os abrir e não vir nada, provavelmente estão incorretos!) Execute o comando abaixo para excluir todos os arquivos JPEG no seu diretório de trabalho atual.

```
rm *.jpg
```

Se preferir não ser solicitado a confirmar cada exclusão, execute o comando abaixo.

```
rm -f *.jpg
```

Tenha cuidado com essa `-f` opção, pois ela "força" a exclusão sem avisar.

Passo a passo

Como testar

Executando o Programa

```
./recover card.raw
```

Correção

```
check50 cs50/problems/2025/x/recover
```

Estilo

```
style50 recover.c
```

Como enviar

```
submit50 cs50/problems/2025/x/recover
```