Disciplina: Programação Estruturada
Turmas: A1 e A2 – Noturno

Prof. Dr. Jesús P. Mena-Chalco
Assistente Docente: Rafael J. P. Damaceno

Universidade Federal do ABC

Estrutura e Arquivos

Neste tutorial serão trabalhados os seguintes tópicos:

1. Definição e uso de estruturas (`struct`)
2. Abertura, leitura e escrita de arquivos
3. Abertura, leitura e escrita de arquivos de imagem bitmap (.bmp)
4. Aplicação de filtro em arquivos de imagem

1. Definição e uso de estruturas

Estrutura (`struct`) é um tipo de dado definido pelo usuário. É útil para agrupar em um mesmo elemento os atributos de um objeto. Por exemplo, considere um ponto no plano cartesiano, composto pelas coordenadas x e y. Pode-se definir uma estrutura chamada `coordenada_cartesiana` em que cada atributo é uma coordenada. Para isso, usa-se a palavra-chave `struct` seguida do nome da estrutura. E no escopo interno declaram-se as variáveis (atributos) que a estrutura contém.

Exemplo 1. Estrutura que representa um ponto no plano cartesiano.

```
#include <stdio.h>
int main(){
    // definição da estrutura coordenada_cartesiana
    struct coordenada_cartesiana {
        float x;
        float y;
    };

    // instanciação da variável ponto1 como uma coordenada cartesiana
    struct coordenada_cartesiana ponto1;
    ponto1.x = 13.4; // atribuição do valor 13.4 à coordenada x do ponto1
    ponto1.y = 1.6;  // atribuição do valor 1.6 à coordenada y do ponto1

    struct coordenada_cartesiana ponto2;
    ponto2.x = 2.3;
    ponto2.y = 2.1;

    printf('Soma das coordenadas x dos pontos 1 e 2: %.1f', ponto1.x + ponto2.x);
 return 0;
}
```

Exercício 1. Modifique o exemplo 1 para conter três coordenadas do tipo inteiro, isto é, cada ponto é composto pelos valores inteiros x, y e z.
Exercício 2. Crie um vetor composto por 10 elementos da estrutura definida no exercício 1. Atribua a cada coordenada um valor randômico. Imprima a soma das coordenadas z dos 10 pontos.
Exercício 3. Crie uma função para calcular a distância euclidiana entre dois pontos. Teste a função com dois pontos do exercício 2.
Exercício 4. Defina uma estrutura chamada `horario`, composta por hora, minuto e segundo.
Exercício 5. Crie uma função chamada `diferenca_de_horarios(struct horario inicio, struct horario fim)` para retornar a diferença entre dois horários (em segundos), com base na estrutura definida no exercício anterior. Por exemplo, para inicio = 20:05:10 (20 horas, 5 minutos e 10 segundos) e fim = 20:35:15, o resultado é: 1805 (segundos).

2. Abertura, leitura e escrita de arquivos

Arquivo é uma forma de manter os dados de um programa salvos, mesmo após seu encerramento. Também é útil para carregar dados externos em determinados programas. Em linguagem C, arquivos são manipulados por meio de um ponteiro para `FILE`. Para abrir arquivos, usa-se a função `fopen(char *nome_do_arquivo, char modo)`, em que `nome_do_arquivo` é o camanho do arquivo que se quer abrir, e `modo` é a forma de abertura que se quer realizar (conforme Tabela 1). Para fechar um arquivo, usa-se a função `fclose(FILE *fp)`, onde `FILE *fp` é o ponteiro para o arquivo.

Modo	Descrição	Tipo de arquivo
"r"	Abre arquivo somente para leitura	Texto
"w"	Abre arquivo novo para escrita (caso já exista, é sobrescrito)	Texto
"a"	Abre arquivo para escrita a partir da última linha	Texto
"rb"	Abre arquivo somente para leitura	Binário
"wb"	Abre arquivo novo para escrita (caso já exista, é sobrescrito)	Binário
"ab"	Abre arquivo para escrita a partir da última linha	Binário

Tabela 1: Modos de abertura de arquivo na linguagem C.

Existem diversas funções em C para escrita e leitura de dados em arquivos. Para escrever dados em arquivos de texto pode-se utilizar, por exemplo, as funções `fprintf()`, `fputs()`, `putc()` e `fputc()`. Para ler dados de arquivos de texto, por exemplo, pode-se utilizar a função `fscanf()`. Os Exemplos 2 e 3 ilustram o uso de algumas destas funções.

Exemplo 2. Escrita de dados em arquivos de texto.

```
1  #include <stdio.h>
2  int main(){
3      FILE *parq;                      // declaração de ponteiro para FILE
4      parq = fopen("dados.txt", "w"); // abertura de arquivo no modo escrita
5      if (parq == NULL)                // caso não seja possível abrir arquivo
6         return 1;
7      fputs("ola mundo usando fputs\n", parq);      // escrita de texto no arquivo
8      fprintf(parq, "ola mundo usando fprintf\n"); // escrita de texto no arquivo
9      fputc('A', parq);                             // escrita de caractere no arquivo
10     fclose(parq);                    // fechamento de arquivo
11     return(0);
12 }
```

Exemplo 3. Leitura de dados a partir de arquivos de texto.

```
1  #include <stdio.h>
2  int main(){
3      FILE *parq;                      // declaração de ponteiro para FILE
4      parq = fopen("dados.txt", "r"); // abertura de arquivo no modo leitura
5      if (parq == NULL)                // caso não seja possível abrir arquivo
6         return 1;
7      char c;
8      // enquanto houver algo a ser lido, leia
9      while(fscanf(parq, "%c", &c) == 1)
10         printf("%c", c);             // imprima o caractere lido
11     printf("\n");
12     fclose(parq);                    // fechamento de arquivo
13     return(0);
14 }
```

Também é possível ler e gravar estruturas (`structs`) inteiras em arquivos, de uma só vez. Para isso, pode-se utilizar as funções `fread()` e `fwrite()`, para ler e escrever, respectivamente. Os Exemplos 4 e 5 ilustram o uso destas funções. No Exemplo 3, as linhas 21-22 gravam no arquivo "coordenadas.dat" os pontos 1 e 2 declarados nas linhas 3-13. O primeiro argumento de fwrite() é um *buffer* com o conteúdo a ser gravado (o endereço do conteúdo). O segundo é o tamanho do tipo de dado a ser gravado, no caso, o tamanho da estrutura `coordenada_cartesiana`. O terceiro argumento representa a quantidade de estruturas que serão gravadas na chamada da função e o último é o ponteiro para o arquivo. Os mesmos argumentos são utilizados para a função `fread`, com o diferencial de que o primeiro argumento é o endereço da variável em que serão armazenados os `structs` lidos. Pode, por exemplo, ser um vetor, como ilustra a linha 16 do Exemplo 4.

Exemplo 4. Escrita em arquivo de dados do tipo `struct`.

```
1  #include <stdio.h>
2  int main(){
3      struct coordenada_cartesiana {
4          float x;
5          float y;
6      };
7      struct coordenada_cartesiana ponto1;
8      ponto1.x = 13.4;
9      ponto1.y = 1.6;
10
11     struct coordenada_cartesiana ponto2;
12     ponto2.x = 2.3;
13     ponto2.y = 1.1;
14
15     FILE *parq;                              // declaração de ponteiro para FILE
16     parq = fopen("coordenadas.dat", "wb"); // abertura de arquivo no modo escrita
17     if (parq == NULL)                        // caso não seja possível abrir arquivo
18         return 1;
19
20     // fwrite(buffer, tamanho, quantidade, arquivo)
21     fwrite(&ponto1, sizeof(struct coordenada_cartesiana), 1, parq);
22     fwrite(&ponto2, sizeof(struct coordenada_cartesiana), 1, parq);
23  return 0;
24 }
```

Exemplo 5. Leitura de arquivos contendo dados do tipo `struct`.

```
1  #include <stdio.h>
2  int main(){
3      struct coordenada_cartesiana {
4          float x;
5          float y;
6      };
7      // declaração de vetor de coordenada_cartesiana
8      struct coordenada_cartesiana pontos[2];
9
10     FILE *parq;                              // declaração de ponteiro para FILE
11     parq = fopen("coordenadas.dat", "rb"); // abertura de arquivo no modo escrita
12     if (parq == NULL)                        // caso não seja possível abrir arquivo
13         return 1;
14
15     // fread(buffer, tamanho, quantidade, arquivo)
16     fread(pontos, sizeof(struct coordenada_cartesiana), 2, parq);
17
18     for (int i = 0; i < 2; i++)
19         printf("(%.1f,%.1f)\n", (*(pontos + i)).x, (*(pontos + i)).y);
20  return 0;
21 }
```

Exercício 6. No Exemplo 5, experimente trocar o terceiro argumento da função fread() (linha 16) para 1. Nesse caso, apenas 1 ponto será lido. O que imprime a linha correspondente ao segundo ponto?
Exercício 7. Altere os Exemplos 4 e 5 para escrever e ler a estrutura do Exercício 1 (ponto com três coordenadas).
Exercício 8. Altere os Exemplos 4 e 5 para escrever e ler a estrutura do Exercício 4 (horário com hora, minuto e segundo).

3. Abertura, leitura e escrita de arquivos de imagem

Arquivos de imagem são arquivos binários que possuem, geralmente, múltiplas estruturas pré-definidas. O formato bitmap (.bmp), por exemplo, é composto por três principais partes, a saber, cabeçalhos, índice de cores e dados de pixels. Os cabeçalhos possuem uma série de informações sobre o formato em si e sobre uma imagem em específico, tais como o tamanho do arquivo, o comprimento, a altura, o tipo de compressão utilizada, entre outras. No índice de cores estão definidas as cores e nos dados de pixels estão as informações de cada pixel, i.e., o valor de cada pixel considerando os tons de *red*, *green* e *blue*. É possível ler uma imagem em linguagem C, aplicar alguma modificação nos pixels e gravar em um novo arquivo a versão modificada, por exemplo. O Exemplo 6 ilustra como ler, modificar e gravar imagens no formato bitmap.

Exemplo 6. Abertura, leitura e gravação de arquivos bitmap.

```c
#include <stdio.h>
#include <stdlib.h>
// define cabeçalho do formato bitmap
typedef struct file_header {
    unsigned short file_type;
    unsigned int file_size;
    unsigned short reserved1;
    unsigned short reserved2;
    unsigned int offset;
} file_header;
// define cabeçalho do arquivo em questão
typedef struct info_header {
    unsigned int info_header_size;
    unsigned int width;
    unsigned int height;
    unsigned short planes;
    unsigned short bits;
    unsigned int compression;
    unsigned int size;
    unsigned int xdata_pixels_per_meter;
    unsigned int ydata_pixels_per_meter;
    unsigned int colors_used;
    unsigned int important_colors;
} info_header;

int main(){
    FILE *pointer_to_image;
    pointer_to_image = fopen("example.bmp", "rb");
    if (pointer_to_image == NULL)
        return 1;
    file_header header;
    info_header info;
    fread(&header, 14, 1, pointer_to_image); // lê 14 bytes para cabeçalho
    fread(&info, 40, 1, pointer_to_image);   // lê 40 bytes para info
    // instancia e aloca memória para dados de pixels
    unsigned char *data_pixels = (unsigned char*) malloc(info.size);
    // lê pixels da imagem para variável data_pixels
    fread(data_pixels, info.size, 1, pointer_to_image);
    fclose(pointer_to_image);
    FILE *pointer_to_new_image;
    pointer_to_new_image = fopen("new_example.bmp", "wb");
    // grava cabeçalhos da imagem nova
    fwrite(&header, 14, 1, pointer_to_new_image);
    fwrite(&info, 40, 1, pointer_to_new_image);
    // pinta todos os pixels de verde
    for (int i = 0; i < info.size; i+=3){
        data_pixels[i] = 0;  // blue
        data_pixels[i+1] = 255; // green
        data_pixels[i+2] = 0;  // red
    }
    // grava data_pixels na imagem nova
    fwrite(data_pixels, info.size, 1, pointer_to_new_image);
    fclose(pointer_to_new_image);
    return 0;
}
```

4. Aplicação de filtro em arquivos de imagem