Arquivos Binário

Algoritmos e Programação 2

Prof. Dr. Anderson Bessa da Costa
Universidade Federal de Mato Grosso do Sul

Arquivos Binário em C/C++

- Arquivos em C/C++ não podem ser associados a tipo primitivo ou registro
  - Armazenam sequência de caracteres ou de *bytes*
- Em vários momentos, é útil:
  - Ler de um arquivo e gravar diretamente em variável **int** ou **float**, ou, ainda em uma variável **struct**
  - Gravar o conteúdo de uma variável em um arquivo

Arquivos Binário em C/C++ (cont.)

- Isto é **arquivo binário**
  - Uma operação de leitura ou escrita deverá ser informar o número de *bytes* que serão lidos ou gravados
  - Função **sizeof()** nos auxiliará

Ler e Escrever em Arquivo Binário

- **fwrite**: transfere *bytes* da memória para o arquivo
- **fread**: transfere *bytes* do arquivo para a memória

Problema Salvar Vetor

Suponha que você tenha um vetor de inteiros de 10 elementos, e gostaria de salvar em um arquivo esse vetor para posteriormente carregar do arquivo.

Solução com Arquivo Texto (1)

```c
#include <stdio.h>

int main() {
  int v[] = {5, 3, 2, 1, 4, 6, 7, 8, 9};
  FILE *f;

  // abra o arquivo
  f = fopen("vetor.txt", "w");

  // se arquivo nao foi aberto corretamente
  if (f == NULL) {
    // imprima mensagem de erro e finalize
    printf("Erro ao abrir arquivo!\n");
    return 1;
  }

  // para cada elemento do vetor
  for (i = 0; i < 9; i++) {
    // escreva elemento no arquivo
    fprintf(f, "%d ", v[i]);
  }

  // feche o arquivo
  fclose(f);
  return 0;
}
```

Solução com Arquivo Texto (2)

Solução com Arquivo Texto (3)

```c
#include <stdio.h>

int main() {
  int v[9], i, n = 0, nread;
  FILE *f;

  f = fopen("vetor.txt", "r");
  if (f == NULL) {
    printf("Erro ao abrir arquivo!\n");
    return 1;
  }

  // carregue elementos do arquivo para o vetor
  while(!feof(f)) {
    nread = fscanf(f, "%d", &v[n]);
    if (nread <= 0)
      break;
    n = n + 1;
  }

  // imprima vetor
  for (i = 0; i < n; i++) {
    // imprima elemento na tela
    printf("%d ", v[i]);
  }

  fclose(f);
  return 0;
}
```

```
5 3 2 1 4 6 7 8 9
```

Solução com Arquivo Binário (1)

```c
#include <stdio.h>

int main() {
  int v[] = {5, 3, 2, 1, 4, 6, 7, 8, 9}, i;
  FILE *f;

  // abra o arquivo
  f = fopen("vetor.bin", "wb");

  // se arquivo nao foi aberto corretamente
  if (f == NULL) {
    // imprima mensagem de erro e finalize
    printf("Erro ao abrir arquivo!\n");
    return 1;
  }

  // escreva bytes no arquivo
  fwrite(v, sizeof(int), 9, f);

  // feche o arquivo
  fclose(f);
  return 0;
}
```

Solução com Arquivo Binário (2)

```c
#include <stdio.h>

int main() {
  int v[9], i;
  FILE *f;

  f = fopen("vetor.bin", "rb");
  if (f == NULL) {
    printf("Erro ao abrir arquivo!\n");
    return 1;
  }

  // carregue bytes do arquivo para o vetor
  fread(v, sizeof(int), 9, f);

  // imprima vetor
  for (i = 0; i < 9; i++) {
    // imprima elemento na tela
    printf("%d ", v[i]);
  }

  fclose(f);
  return 0;
}
```

```
5 3 2 1 4 6 7 8 9
```

Função fread

fread

<cstdio>

```
size_t fread ( void * ptr, size_t size, size_t count, FILE * stream );
```

Read block of data from stream

Reads an array of *count* elements, each one with a size of *size* bytes, from the *stream* and stores them in the block of memory specified by *ptr*.

The position indicator of the stream is advanced by the total amount of bytes read.

The total amount of bytes read if successful is `(size*count)`.

Parameters

`ptr`
: Pointer to a block of memory with a size of at least `(size*count)` bytes, converted to a `void*`.

`size`
: Size, in bytes, of each element to be read.
size_t is an unsigned integral type.

`count`
: Number of elements, each one with a size of *size* bytes.
size_t is an unsigned integral type.

`stream`
: Pointer to a FILE object that specifies an input stream.

Função fwrite

fwrite

<cstdio>

```
size_t fwrite ( const void * ptr, size_t size, size_t count, FILE * stream );
```

Write block of data to stream

Writes an array of *count* elements, each one with a size of *size* bytes, from the block of memory pointed by *ptr* to the current position in the *stream*.

The *position indicator* of the stream is advanced by the total number of bytes written.

Internally, the function interprets the block pointed by *ptr* as if it was an array of `(size*count)` elements of type `unsigned char`, and writes them sequentially to *stream* as if `fputc` was called for each byte.

Exemplo 1: Crie arquivo binário

```c
#include <stdio.h>

struct clientData {
  int acctNum;
  char lastName[15];
  char firstName[10];
  double balance;
};

int main () {
  int i;
  struct clientData blankClient = {0, "", "", 0.0};
  FILE *f;

  // abra o arquivo no modo escrita binario
  f = fopen("credit.dat", "wb");
  // se arquivo nao foi aberto corretamente
  if (f == NULL) {
    // imprima mensagem de erro e finalize o programa
    printf("Arquivo nao pode ser aberto!\n");
    return 1;
  }
  // repita 100 vezes
  for(i = 1; i <= 100; i++)
    // escreva no arquivo os bytes de um registro com os valores default
    fwrite(&blankClient, sizeof(struct clientData), 1, f);
  // feche o arquivo
  fclose(f);
  return 0;
}
```

Função fseek

fseek

<cstdio>

```
int fseek ( FILE * stream, long int offset, int origin );
```

Reposition stream position indicator

Sets the position indicator associated with the *stream* to a new position.

For streams open in binary mode, the new position is defined by adding *offset* to a reference position specified by *origin*.

For streams open in text mode, *offset* shall either be zero or a value returned by a previous call to ftell, and *origin* shall necessarily be `SEEK_SET`.

If the function is called with other values for these arguments, support depends on the particular system and library implementation (non-portable).

The *end-of-file internal indicator* of the *stream* is cleared after a successful call to this function, and all effects from previous calls to ungetc on this *stream* are dropped.

On streams open for update (read+write), a call to `fseek` allows to switch between reading and writing.

Função fseek (cont.)

Constant	Reference position
SEEK_SET	Beginning of file
SEEK_CUR	Current position of the file pointer
SEEK_END	End of file *

Constantes utilizadas para posicionar o ponteiro do arquivo.

Exemplo 2: Escreva no arquivo

```
...

int main () {
    struct clientData client;
    FILE *f;

    f = fopen("credit.dat", "rb+");
    if (f == NULL) {
        printf("Arquivo nao pode ser aberto!\n");
        return 1;
    }
    printf("Enter account number (1 to 100, 0 to end input)\n?");
    scanf("%d", &client.accNum);
    while (client.acctNum != 0) {
        printf("Enter lastname, firstname, balance\n?");
        scanf("%s%s%f", client.lastName, client.firstName, &client.balance);
        fseek(f, (client.accNum - 1) * sizeof(struct clientData), SEEK_SET);
        fwrite(&client, sizeof(struct clientData), 1, f);
        printf("Enter account number\n? ");
        scanf("%d", &client.acctNum);
    }
    fclose(f);
    return 0;
}
```

Exemplo 2: Escreva no arquivo (cont.)

```
Enter account number (1 to 100, 0 to end input)
? 37
Enter lastname, firstname, balance
? Barker Doug 0.00
Enter account number
? 29
Enter lastname, firstname, balance
? Brown Nancy -24.54
Enter account number
? 96
Enter lastname, firstname, balance
? Stone Sam 34.98
Enter lastname, firstname, balance
? 0
```

Exemplo 3: Leia do arquivo

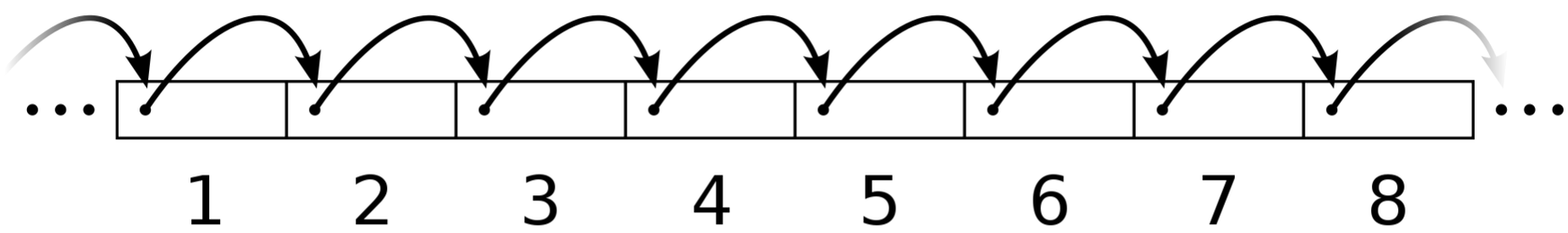
```
...
int main () {
  int i, nread;
  struct clientData client;
  FILE *f;

  f = fopen("credit.dat", "rb");
  if (f == NULL) {
    printf("Arquivo nao pode ser aberto!\n");
    return 1;
  }
  printf("%-6s%-16s%-11s%10s\n", "Acct", "Last Name", "First Name", "Balance");
  while (!feof(f)) {
     nread = fread(&client, sizeof(struct clientData), 1, f);
     if (nread <= 0)
       break;
     if (client.accNum != 0) {
       printf("%-6d%-16s%-11s%10.2f\n",
         client.accNum, client.lastName,
         client.firstName, client.balance);
     }
  }
  fclose(f);
  return 0;
}
```

Exemplo 3: Leia do arquivo (cont.)

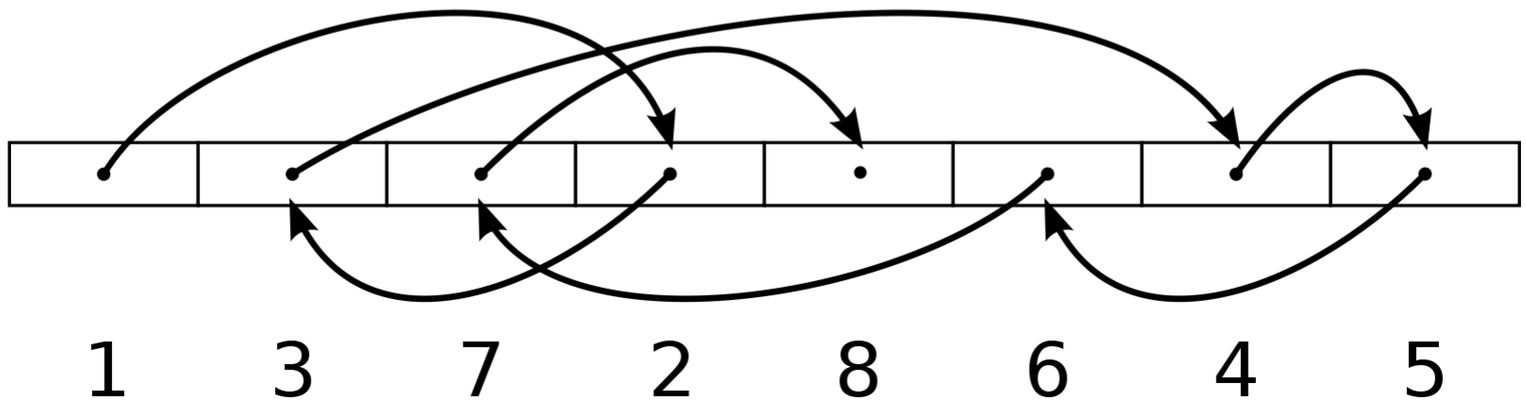
```
Acct  Last Name       First Name    Balance
29    Brown           Nancy          -24.54
33    Dunn            Stacey         314.33
37    Barker          Doug             0.00
88    Smith           Dave           258.34
96    Stone           Sam             34.98
```

Acesso Sequencial vs Acesso Aleatório

Sequential access (Arquivos Texto)



Random access (Arquivos Binário)



Por Que Acesso Sequencial?

- Não podem ser modificados sem risco de destruir outros dados
  - Campos podem variar em tamanho
  - Diferentes representações em arquivos e tela do que representações internas
  - 1, 34, -890 são todos inteiros, mas possuem tamanhos diferentes no disco

Dados Podem Ser Perdidos

```
300 White 0.00 400 Jones 32.87   (old data in file)
```

If we want to change White's name to Worthington,

```
300 Worthington 0.00
```

```
300 White 0.00 400 Jones 32.87
```

Data gets overwritten

```
300 Worthington 0.00ones 32.87
```

Referências

- ASCENCIO, A. F. G.; CAMPOS, E. A. V. Fundamentos da programação de computadores: algoritmos, Pascal, C/C ++ e Java. 3. ed. São Paulo: Pearson, 2012.
- DEITEL, P.; DEITEL, H. C: Como Programar. 6ª ed. São Paulo: Pearson, 2011.
- PIVA, D.J. et al. Algoritmos e programação de computadores. Rio de Janeiro: Elsevier, 2012.