Programação em C++:
Entrada e saída de dados

J. Barbosa

J. Tavares

Biblioteca de *Streams* do C++

- As **acções de entrada** e saída de dados **não fazem parte da linguagem C++.**
- Por forma a uniformizar as primitivas através das quais um programa invoca as acções de I/O (entrada e saída de dados) a linguagem *C++* virtualiza todos os dispositivos envolvidos nestas acções como objectos *streams*.
- A linguagem *C++* dispõe de uma biblioteca de classes *stream,* cujas declarações se situam nos ficheiros iostream.h, iomanip.h e fstream.h, satisfazendo ao paradigma da **Programação Orientada por Objectos**

Biblioteca de *Streams* do C++

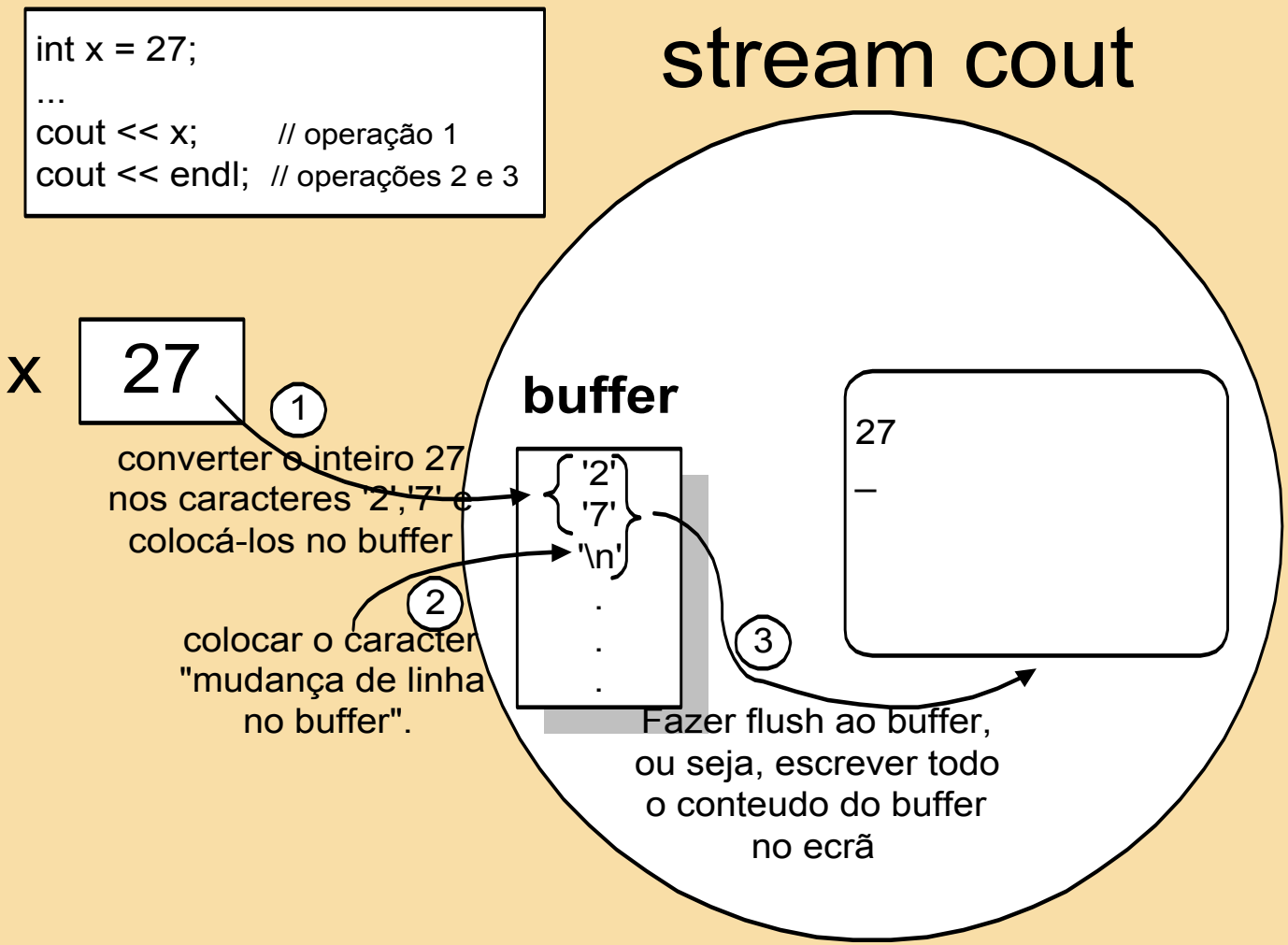
- Todos os **dispositivos lógicos** (*streams*) são semelhantes em comportamento, e bastante **independentes dos dispositivos reais**.
- Distinguem-se **dois tipos de *streams*** - ***streams* para texto** e ***streams* para palavras binárias**
- Um *stream*, associa-se a um periférico realizando uma operação abertura (**open**), e desassocia-se dele com uma operação de fecho (**close**).

cin e *cout*

- As *streams* **cin** e **cout** tornam-se a interface entre o programa e o utilizador, para interactuar com o ***teclado*** e com o ec**rã**.
- O *stream* **cin** é **criado automaticamente** quando se inclui o ficheiro header **<iostream.h>** ficando associado ao *teclado* do terminal.
- O *stream* **cout** é **criado automaticamente** quando se inclui o ficheiro *header* **<iostream.h>** ficando associado ao ecrã do terminal

Saída de dados - operador << (1)

```
int x = 27;
...
cout << x;        // operação 1
cout << endl;     // operações 2 e 3
```



Saída de dados - operador << (2)

O ***operador insersor*** **<<**, retorna uma **referência** para o **ostream** sobre o qual operou, pelo que se podem concatenar inserções numa mesma expressão.

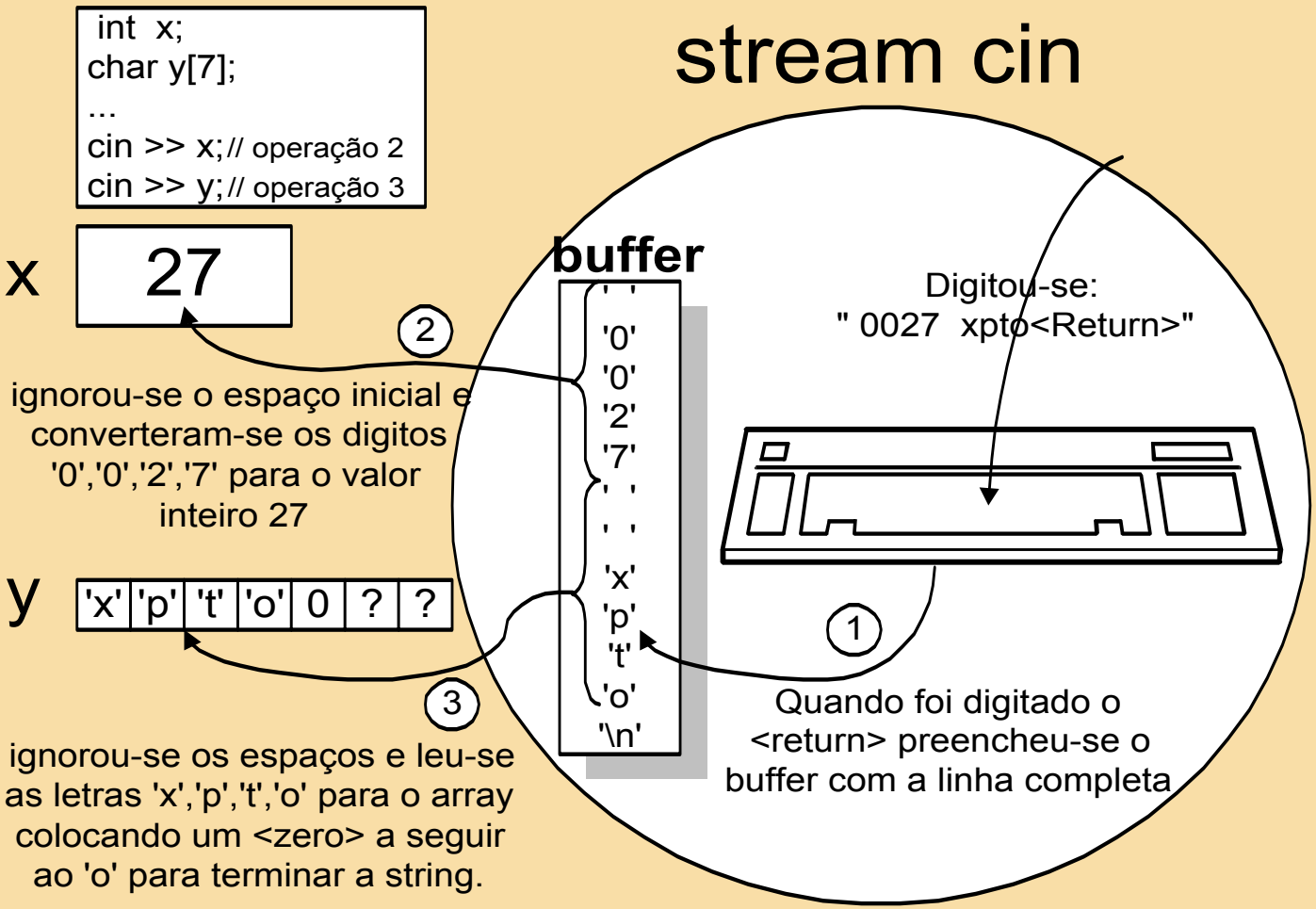
```
void main()
  int i= 10, double d= 3.1456;

  cout <<  "i =" << i << ", d=" << d << endl;

  /* Equivalente  a:
  cout  << "i =";      // insere  a string "i ="
  cout  << i;          // insere  o valor inteiro de i
  cout  << ", d=" ;    // insere a string
  cout  << d;          // insere o valor real de d
  cout  << endl;       // insere o caracteres '\r\n' e
                       //faz flush do buffer.

  */
}
```

Entrada de dados - operador >> (1)



Entrada de dados - operador >> (2)

- O operador **extractor >>,**toma como **operando esquerdo** um ***istream*** e como **operando direito** a **variável a afectar** com o valor extraído do istream.
- O **operando direito** pode ser **qualquer dos tipos intrínseco do C++**, pois em <iostream.h> estão definidos várias versões do operador insersor .
- Por omissão, **salta espaços em branco** (como definido na função **isspace()** em <ctype.h>), lendo seguidamente os caracteres adequados ao tipo de objecto que vai ser afectado.

```
void main() {
   char c;
   int i;
   char str[10];
   float f;

   cin >> c >> i  >> f >> str;

  /* Equivalente  a:
   cin  >> c; // extrai um carácter
              // diferente de separador.
   cin >> i;   // extrai um valor inteiro.
   cin >>f;    // extrai um valor real.
   cin >> str;// extrai uma palavra.
  */

    cout << "c = " << c << endl
         << "i = "  << i  << endl
         << "f = ”  << f  << endl
         << "str = " << str << endl;
}
```

Manipuladores

Existem **manipuladores** para *streams*, de **entrada** (mudam o formato das extracções) e/ou de *saída* (mudam o formato das inserções)

Os manipuladores estão **declarados** em **<iomanip.h>**. Os mais comuns são:

Manipulador	*In*	*Out*	Definição
`endl`		v	Mudar de linha e *flush* do `ostream`.
`ends`		v	Inserir '\0' para terminar *string*.
`flush`		v	Esvaziar (*flush*) o *buffer* do `ostream`.
`dec`	v	v	Conversão para base decimal.
`hex`	v	v	Conversão para base hexadecimal.
`oct`	v	v	Conversão para base octal.
`ws`	v		Eliminar caracteres separadores.
`setbase(int b)`	v	v	Fixar a base de conversão em `b`.
`resetiosflags(long b)`	v	v	Desactivar *bit*-vector *flags* de acordo com `b`. Ver
`setiosflags(long b)`	v	v	Activar os *bit*-vector *flags* de acordo com `b`. Ver
`setfill(int f)`		v	Definir o carácter de preenchimento de espaços do campo com `(char)f`
`setprecision(int n)`		v	Situar em `n` dígitos a precisão de um *floating-point*.
`setw(int n)`	v	v	Colocar em `n` caracteres a largura do campo.

Manipuladores - exemplo

```
#include<iostream.h>
#include<iomanip.h>
void main() {
  int i=123;
  double d=12345.6789;
  char *s= "blabla";
  cout << setw(10) << s << endl
       << i << endl
       << setw(10) << i << endl
       << d << endl
       << setw(15) << setfill('*') << d << endl
       << setw(13) << setfill(' ')
       << setprecision(3)
       << setiosflags(ios::left) << d << s
       << endl;
}
```

Qual o *output* deste programa?

```
    blabla
123
       123
12345.6789
*****12345.6789
12345.679    blabla
```

E o resultado de?

x=65;

cout << x;

Flags de formatação de um *stream*

Flag	*In*	*Out*	Definição
`ios::skipws`	**v**		Salta espaços em branco no *input*.
`ios::left`		**v**	Espaços à esquerda.
`ios::right`		**v**	Espaços à direita.
`ios::internal`		**v**	Espaços entre o sinal e o valor.
`ios::dec`	**v**	**v**	Conversão para base decimal.
`ios::hex`	**v**	**v**	Conversão para base hexadecimal.
`ios::oct`	**v**	**v**	Conversão para base octal.
`ios::fixed`		**v**	Usa a notação 123.45 para os *floating point*.
`ios::scientific`		**v**	Usa a notação 1.2345E2 para os *floating point*.
`ios::showbase`		**v**	Força a escrita da base.
`ios::showpoint`		**v**	Força a escrita do ponto (*floating point*).
`ios:: showpos`		**v**	Adiciona o '+' nos inteiros positivos.
`ios::uppercase`		**v**	Escreve em maiúsculas quando a base é hexadecimal.

Usado nas instruções:
setiosflags()
resetiosflags()

Ou isoladamente: cout << scientific << 10.0;

Métodos de entrada e saída não formatada

get

istream & get(char &ch); // lê um único carácter;

// Ex . - Copiar uma linha do standard *input* para o standard *output*

```
#include <iostream.h>
void main()  {
   char ch;
   do {   cin.get(ch);  cout << ch;   } while (ch!= '\n');
}
```

put

Saída de caracteres sem formatação:

```
int ch='x';
cout  << setw(10);
cout.put(ch);        // mostra o carácter cujo código é ch. Não liga
                     //  às flags

cout << (char)ch; // Mostra o mesmo carácter num campo de 10
                  //  caracteres
```

Saída: x x

Métodos de entrada e saída não formatada

getline

istream::getline(char *line, int size, char terminator)

/*

Os caracteres são extraídos até que uma das seguintes condições se verifiquem:

- **size-1** caracteres sejam lidos;
- não existam mais caracteres para ler (EOF - fim de ficheiro);
- o caracter lido seja o caracter terminador. Neste caso, o caracter é lido mas não é inserido no vector line.

No fim da sequência de caracteres é sempre inserido o caracter nulo (‘\0’), logo a dimensão máxima da string é size-1 caracteres .

Ex . - Digitar uma linha terminada por 'p'

*/

```
#include <iostream.h>
void main()  {
   char line[100];
   cout << " Digite uma linha terminado por 'p'" << endl;
   cin.getline( line, 100, 'p' );// 'p' é caracter terminador
   cout << line;
}
```

Funções para filtrar caracteres

As funções seguintes, reconhecem tipos de caracteres retornando **true** ou **false** conforme o caracter testado satisfizer ou não a condição da função evocada.

#include <ctype.h>

```
int   isdigit(char)     // '0' .. '9'
int   islower(char)    // 'a' .. 'z'
int   isupper(char)     //  'A' .. 'Z'
int   isalpha(char)    // islower() | isupper()
int   isalnum(char)   // isalpha()  | isdigit()
int   isxdigit(char)   // '0' .. '9' 'a'  .. 'f' 'A' .. 'F'
int   isascii(char)     //  0 .. 0x7F
int   iscntrl(char)     //caracteres de controlo
int   isgraph(char)   //isalpha() | isdigit() | ispunct ()
int   isprint(char)     // printable: ascii entre '  ' e '~'
int   ispunct(char)    // pontuação
int   isspace(char)     // ' ' '\t' CR  LF
```

Outras funções de ctype.h

toascii(char) – converte para ASCII

toupper(char) – converte para maiúsculas

tolower(char) – converte para minúsculas

Acesso a ficheiros

- Podem ser definidos objectos associados a ficheiros, e passar a interactuar com esses objectos com os mesmos operadores, métodos e manipuladores que se utilizam para **cin** e **cout**.
- Existem vários objectos que podemos criar para ter acesso a ficheiros:
  - **ifstream** - quando queremos abrir um ficheiro para leitura.
  - **ofstream** - quando queremos abrir um ficheiro para escrita.
  - **fstream** - quando se deseja que o ficheiro possa ser lido e escrito.
- Para criar qualquer um dos tipos de objectos anteriores, teremos de explicitamente proceder aos vários passos da definição, que nos são ocultos no caso de **cin** e **cout**.

Métodos open() e close() sobre streams

Acesso a um ficheiro para leitura.

```
#include <fstream.h>
ifstream is;  // ficheiro de input
// Abrir o ficheiro para ler.
is.open("c:\mensage\text1.doc");
```

Equivale a:

```
ifstream is("c:\mensage\text1.doc");
```

```
is.close(); // Fechar o ficheiro de input
```

Acesso a um ficheiro para escrita.

```
#include <fstream.h>
ofstream os; // ficheiro de output
// Abrir o ficheiro para escrita.
os.open("c:\mensage\text2.doc");
```

Equivale a:

```
ofstream os("c:\mensage\text2.doc");
```

```
os.close(); // Fechar o ficheiro de output
```

O método **close()** garante que toda a informação situada no *buffer* é transferida para ficheiro em disco, e que as estruturas de dados inerentes à organização do disco sejam devidamente actualizadas.

Testar se o *fstream* foi aberto com sucesso

No caso de uma acção de **open**() sobre um **fstream** não ser bem conseguida, por qualquer motivo, esse facto pode ser reconhecido em programa, testando o objecto ***stream*** como valor lógico ou usando o método **fail().**

```
If (f1.fail())
    cout << “Erro”;
```

Programa para copiar do ficheiro “file.in” para “file.out”

```
#include<fstream.h>
#include<iostream.h>
#include <stdlib.h>
void main()
{    char ch;
     ifstream f1("file.in");
     if (!f1){          // Teste ao estado da fstream input
          cout  << “Erro a abrir ficheiro de leitura." << endl;
          exit(0);
     }
     ofstream f2("file.out");
     if (!f2) {         // Teste ao estado da fstream de output
          cout  << “Erro a abrir ficheiro de escrita." << endl;
          exit(0);
     }
     while ( f1.get(ch) ) f2.put(ch);
     f1.close();
     f2.close();  cin.get();
}
```

Modos de acesso de um *fstream*

- Ao contrário de objectos do tipo **ifstream** e **ofstream**, que têm modos de acesso pré- estabelecidos, os objectos do tipo **fstream**, podem ter acesso para escrita, para leitura, ou ambos.
- A iniciação de um objecto **fstream** pode ser efectuada com um único parâmetro *string*, mas também podemos explicitar num segundo parâmetro a especificação de modos alternativos de acesso.

Mode bit	**Acção**
`ios::app`	*Append data* - Escreve no fim do ficheiro
`ios::ate`	Posiciona-se no fim do ficheiro inicialmente
`ios::in`	Abre o ficheiro para leitura
`ios::out`	Abre o ficheiro para escrita
`ios::binary`	Abre o ficheiro em modo binário
`ios::trunc`	Despreza o anterior conteúdo do ficheiro.
`ios::nocreate`	Falha a acção se não existir o ficheiro
`ios::noreplace`	Se o ficheiro existir, falha abertura (***open***) para saída, a menos que **`ate`** ou **`app`** estejam activas.

Características dos *fstreams*

- Todos os operadores, funções e manipuladores usados para **cin** e **cout**, podem, sem nenhuma alteração ser aplicados a ficheiros abertos em modo de texto para leitura ou para escrita.
- Os ficheiros são úteis para registar e obter grandes quantidades de dados.
- Os operadores << e >> estão vocacionadas para ficheiros de texto.
- Para ficheiros binários são usados os métodos ***get(), put(), read() e write()***

Modos de abertura - exemplos (1)

Por omissão do segundo parâmetro, um **ifstream** é aberto no modo leitura de texto e um **ofstream** é aberto no modo escrita de texto.

Abertura para leitura com **ifstream** e para escrita com **ofstream**

```
ifstream in ("t1.doc");        // fstream in("t1.doc", ios::in);
ofstream out("t2.doc");        // fstrteam out("t2.doc", ios::out | ios::trunc);
```

Abertura para leitura e escrita com **fstream**

```
fstream inOut("t1.doc", ios:in | ios::out);                  // texto
fstream inOut("t1.doc", ios:in | ios::out | ios::binary);// binário
```

Abertura de ficheiros em modo binário

```
ifstream in ("t1", ios::binary);
ofstream out("t2", ios::binary);
```

Modos de abertura - exemplos (2)

Abertura para escrita com posicionamento no fim do ficheiro.

```
// fich.  texto com escrita no fim
ofstream out("t1.doc", ios:app);

// fich. binário com escrita no fim
ofstream out("t1.doc", ios:app | ios::binary);

// fich. texto com posicionamento no fim e com possibilidade de acesso directo
ofstream out ("t1.doc", ios:ate);

// fich. binário com posicionamento no fim e com possibilidade de acesso directo
ofstream out ("t1.doc", ios:ate | ios::binary);
```

Escrita em modo **binário** : *write*

write

```
ostream& write(const char*  ptr, int n);
ostream& write(const signed char*  ptr, int n);
ostream& write(const unsigned char* ptr,int n);
```

- Insere no *stream* **n** caracteres. Se o ficheiro for de texto virtualiza o carácter '\n' , num par ('\r', '\n').
- É vocacionada para registar estruturas de dados em ficheiros binários, embora possa ser usada em **ostream** na generalidade.

Armazenar uma data em ficheiro

```
#include <fstream.h>
struct Date { int mo, da, yr; }
void main() {
   Date dt = { 6, 10, 91 };
   ofstream tfile( "date.dat" , ios::binary );
   tfile.write( (char *) &dt, sizeof dt );
}
```

Escrita em modo **binário** : *read*

read

```
ostream&  read(char*   ptr, int n);
ostream&  read(signed char*   ptr, int n);
ostream&  read(unsigned char* ptr, int  n);
```

O método **read()** é vocacionada para a leitura de estruturas de dados de ficheiros binários, embora possa ser usada noutro tipo de *istreams*.

Teste do método read()

```
#include <iostream.h>
#include <fstream.h>
void main()
{    struct {
            double salary;  char name[23];
     } employee;
     ifstream is( "payroll", ios::binary | ios::nocreate );
      if( is ) {  // ios::operator void*()
            is.read( (char *) &employee, sizeof( employee ) );
            cout << employee.name << ' '
            <<  employee.salary << endl;
       }
       else cout << "ERROR: Cannot open file 'payroll'." <<
      endl;
}
```

Outros métodos de ***fstream***

get(char)	Obtém o próximo carácter
getline(string v, int n, '\n')	Obtém caracteres até n-1 ou até encontrar *newline*. Acrescenta '\0'.
peek(char)	Obtém o próximo carácter sem o retirar do *stream*
put(char)	Coloca um carácter no stream
putback(char)	Coloca o carácter no *input stream*
eof(void)	Retorna *true* se pretendemos ler um carácter depois do EOF (end-of-file)
ignore(int n)	Avança n caracteres. Por defeito n=1.

Acesso aleatório a ficheiros

```
fstream farray("Array", ios:: in | ios::out| ios::binary);
```

- O ficheiro criado pode ser utilizado para virtualizar em disco um *array* com acesso por índice para ler e escrever em qualquer dos seus elementos.
- O método **seekg(long n),** posiciona no *byte* **n** (a contar do inicio do ficheiro), o que permite alterar o acesso ao ficheiro para acções de leitura (***get***).
- O método **seekp(long n),** posiciona no *byte* **n** para escrita**(*put*)**, o que permite alterar o acesso ao ficheiro para acções de escrita.
- **tellg(void)** : retorna o valor do apontador do ficheiro de leitura
- **tellp(void)** : retorna o valor do apontador do ficheiro de escrita

Exemplo – Escrita em ficheiro

```
#include <fstream.h>
#include <stdlib.h>
#include <iomanip.h>

const int MAXLENGTH = 21;  // maximum file name length
char filename[MAXLENGTH] = "test.dat";  // put the filename up front

int main()
{       ofstream out_file;
        out_file.open(filename);

        if (out_file.fail())
        {           cout << "The file was not successfully opened" << endl;
                    exit(1);
        }

        // set the output file stream formats
        out_file << setiosflags(ios::fixed)
                 << setiosflags(ios::showpoint)
                 << setprecision(2);

        // send data to the file
        out_file << "Batteries " << 39.95 << endl
                 << "Bulbs "  << 3.22 << endl
                 << "Fuses " << 1.00;

        out_file.close();
        return 0;
}
```

Exemplo – Leitura do ficheiro (1)

```
#include <fstream.h>
#include <stdlib.h>
#include <iomanip.h>

const int MAXLENGTH = 21;   // maximum file name length
char int MAXCHARS = 31;     // maximum description length
char filename[MAXLENGTH] = "test.dat";


int main()
{
        int ch;

        char descrip[MAXCHARS];
        float price;
        ifstream in_file;

        in_file.open(filename,ios::nocreate);

        if (in_file.fail())  // check for successful open
        {
                cout << "\nThe file was not successfully opened"
                << "\n Please check that the file currently exists."
                << endl;
                exit(1);
        }
```

Exemplo – Leitura do ficheiro (2)

```
    // set the format for the standard output stream
    cout << setiosflags(ios::fixed)
         << setiosflags(ios::showpoint)
         << setprecision(2);

    cout << endl;  // start on a new line

    // read and display the file's contents
    while ( (ch = in_file.peek()) != EOF ) // check next character
    {
            in_file >> descrip >> price;  // input the data
            cout << descrip << ' ' << price << endl;
    }

    in_file.close();

    return 0;
}
```

Exercício

- A função gausseana ou normal é dada pela seguinte equação:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-m)^2}{2\sigma^2}}$$ representada por $N(m, \sigma)$

1. Escreva um programa que calcule e registe em ficheiro de texto os valores da função no intervalo [-a,a] com a resolução da variável x definida pelo utilizador.
2. Acrescente ao programa anterior uma função para leitura do ficheiro e que obtenha a média dos valores.

Nota: O programa deve perguntar se o utilizador pretende calcular os dados (ponto 1) ou ler a partir de ficheiro existente (ponto 2).