Entrada e Saída de Dados

stream

- Uma `stream` (fluxo) é uma abstração que representa um dispositivo no qual as operações de entrada e saída são executadas.
- A `stream` pode ser vista como uma fonte ou destino de caracteres de comprimento indefinido.
- Principais `stream`'s em C++:
  - `cin` – objeto global que representa a `stream` padrão de entrada (geralmente teclado).
  - `cout`, `cerr` – objetos globais que representam as `stream`'s padrão de saída e de emissão de erros (geralmente, ambas são a tela).
  - `ifstream`, `ofstream` – classes para criar `stream`'s de leitura e escrita em arquivos.
  - `istringstream`, `ostringstream` –leitura e escrita em `string`'s.

Métodos de leitura

Principais métodos de leitura de dados a partir de stream (pode ser teclado ou arquivo):

- `operator>>`
  - Inicialmente, lê da `stream` e ignora todos os caracteres delimitadores: espaço, ENTER, TAB.
  - Em seguida, lê e considera todos os caracteres válidos para o tipo de dado em questão.
  - Encerra a leitura ao encontrar o primeiro caractere que não seja válido para o tipo de dado em questão, que pode ser um delimitador ou outro caractere (por exemplo, uma letra para um `int`). O caractere inválido **não é lido**: permanece na `stream` para ser lido na próxima leitura.
  - Geralmente é utilizado para ler dados com tamanho fixo de bytes (`int`, `float`, classes de tamanho fixo, etc.)
  - Não pode ser utilizado para ler strings ou arrays de caracteres, a não ser que o texto não contenha espaços, pois o espaço é um delimitador e encerraria a entrada de dados.
  - Exemplo:
```
#include <iostream>
#include <string>
string S;
cin >> S; // S não pode conter espaços
```
- `getline(` <istream> , <string C++> , <delimitador>`)`
  - Versão do `getline` utilizada para ler strings C++.
  - **Não** ignora eventuais caracteres delimitadores iniciais que estejam no buffer.
  - Lê todos os caracteres que não sejam o delimitador definido na chamada da função (que pode ou não ser algum dos delimitadores padrão: espaço, ENTER, TAB, etc.).
  - Encerra a leitura ao encontrar a primeira ocorrência do caractere delimitador. O caractere delimitador **é lido**, mas é descartado: não é acrescentado à string nem estará disponível para a próxima leitura.
  - O caractere delimitador pode ser qualquer um: caso não seja passado como o terceiro parâmetro para a função, assume-se ENTER (`'\n'`).
  - Exemplo:
```
#include <iostream>
#include <string>
string S;
getline(cin, S, '\n'); // S pode conter espaços
```
- <istream>.`getline(`<array de char C> , <tamanho> , <delimitador>`)`
  - Versão do `getline` utilizada para ler arrays de char ("strings") C.
  - **Não** ignora eventuais caracteres delimitadores iniciais que estejam no buffer.

- Lê todos os caracteres que não sejam o delimitador definido na chamada da função (que pode ou não ser algum dos delimitadores padrão: espaço, ENTER, TAB, etc.).
- Encerra a leitura ao encontrar a primeira ocorrência do caractere delimitador ou quando ler <tamanho-1> caracteres, o que ocorrer primeiro. Caso encontre o delimitador, ele **é lido**, mas descartado: não é acrescentado ao array nem estará disponível para a próxima leitura.
- Acrescenta um caractere zero (`'\0'`) ao final do array de char.
- O caractere delimitador pode ser qualquer um: caso não seja passado como o terceiro parâmetro para o método, assume-se ENTER (`'\n'`).
- Exemplo:

```
#include <iostream>
char S[80];
cin.getline(S, 80, '\n'); // S pode conter espaços; máx 79 char
```

- <istream>.`get`(<array de char C> , <tamanho> , <delimitador>)
  - `get` funciona de maneira praticamente idêntica a <istream>.`getline`. Também é utilizada para ler arrays de char C. A diferença é que, caso encontre o caractere delimitador, a leitura é encerrada, mas o caractere **não é lido**: permanece na stream para ser lido na próxima leitura.
  - Exemplo:

```
#include <iostream>
char S[80];
cin.get(S, 80, '\n'); // S pode conter espaços; máx 79 char
```

Possíveis ordens de execução dos métodos de leitura:

- `operator>>` seguido de outro `operator>>`
  - OK: o segundo `operator>>` começa descartando o ENTER que sobrou da entrada de dados do primeiro `operator>>`.
- `getline` seguido de outro `getline`
  - OK: o primeiro `getline` consome o ENTER final e não deixa nada para o segundo `getline`.
- `operator>>` seguido de um `getline`
  - **ERRO**: o `operator>>` deixa o ENTER final no buffer. Com isso, o `getline` encerrará logo ao ler o primeiro caractere (o ENTER) e não lerá nenhum dos caracteres da string.
- `getline` seguido de um `operator>>`
  - OK: o `getline` consome o ENTER final e não deixa nada para o `operator>>`.

Limpando o buffer:

- Após cada `operator>>` que seja seguido por um `getline`, ou antes de cada `getline` que seja precedido por um `operator>>`, é preciso limpar o buffer da stream para consumir os caracteres ENTER que tenham sobrado da leitura de dados.
- <istream>.`ignore`(<tamanho> , <delimitador>)
  - Lê da `stream` e ignora (descarta) todos os caracteres que não sejam o caractere delimitador, passado como 2º parâmetro da função, descartando no máximo o número de caracteres passado como 1º parâmetro.
  - Caso o valor do 1º parâmetro seja igual a `numeric_limits<streamsize>::max()`, não há limite e serão descartados tantos caracteres quantos necessários até que o caractere delimitador seja encontrado.
  - Exemplo:

```
#include <iostream>
#include <limits>
cin.ignore(numeric_limits<streamsize>::max(), '\n');
```

- <istream> `>> ws`
  - Lê e ignora (descarta) todos os caracteres delimitadores consecutivos a partir da posição atual.

Operações com arquivos (`stream`'s)

`stream`'s de entrada e saída associadas a arquivos:

- Construtor específico:
  - Cria e abre no modo mais usual uma `stream` para leitura ou escrita em arquivo.
  - Exemplo:
```
#include <iostream>
#include <fstream>
ifstream exemplo_in("nome_arquivo_in.dat");    // in
ofstream exemplo_out("nome_arquivo_out.dat"); // out,trunc
exemplo_in >> var;
exemplo_out << var;
```
- <stream>.`open`(<const char* nome> , <modo>)
  - Associa a `stream` com o arquivo cujo nome é passado como 1º parâmetro. O 2º parâmetro determina o modo de abertura (os modos podem ser combinados com |, o OR lógico):
    `in` (input): `stream` aberta para leitura.
    `out` (output): `stream` aberta para escrita.
    `binary`: `stream` faz as operações em binário, e não em modo texto.
    `ate` (at end): a posição de escrita é inicialmente colocada no fim do arquivo.
    `app` (append): todas as operações de saída são feitas no fim do arquivo (acrescentando).
    `trunc` (truncate): descarta todo conteúdo previamente existente no arquivo.
  - Exemplo:
```
#include <iostream>
#include <fstream>
ofstream exemplo_out;
exemplo_out.open("nome_arquivo_out.dat",
                 ifstream::out|ifstream::app);
```
- <stream>.`close()`
  - Fecha o arquivo associado com a `stream`, garantindo que todas as operações de saída pendentes sejam escritas para o arquivo.
  - Exemplo:
```
exemplo_out.close();
```

Teste do status da `stream`:

- <stream>.`eof()` – fim de arquivo atingido
- <stream>.`fail()` – insucesso na última leitura
- <stream>.`good()` – sucesso na última leitura (good = sem `eof` nem `fail` na última leitura)

Posicionando a `stream`:

- <istream>.`seekg(offset, pos)` – posiciona uma stream de entrada (input) a `offset` bytes da posição `pos`. `pos` pode ser `beg` (início), `cur` (posição atual) ou `end` (final).
  Exemplo: `is.seekg (0, is.beg)` - reposiona a stream `is` no início
- <ostream>.`seekp(offset, pos)` – posiciona uma stream de saída (output) a `offset` bytes da posição `pos`. `pos` pode ser `beg` (início), `cur` (posição atual) ou `end` (final).
  Exemplo: `os.seekp (0, os.beg)` - reposiona a stream `os` no início