Linguagens de Programação

Prof. Miguel Elias Mitre Campista

http://www.gta.ufrj.br/~miguel

Parte IV

Introdução à Programação em C++
(Continuação)

Relembrando da Última Aula...

- Template
- Mais exemplos de programação orientada a objetos...

Entrada e Saída de Fluxos de Dados

- I/O em C++
  - Orientado a objeto
    - Utilizam referências, sobrecarga de função, sobrecarga de operador
  - Tipo seguro
    - I/O sensível ao tipo de dados
      - Se alguma função foi elaborada para um tipo de dados específico, ela só será chamada para aquele tipo
      - Erro se tipo não estiver de acordo
  - Definidos pelo usuário e tipos padrão
    - Usuários podem sobrecarregar operadores de I/O (<< ou >>) para tratar de tipos específicos criados
      - Torna o C++ extensível

Fluxos de Dados

- Fluxo de dados: sequência de bytes
  - Entrada:
    - Dos dispositivos (teclado, disco rígido) para memória
  - Saída:
    - Da memória para os dispositivos (monitor, impressora etc.)
  - Aplicações associam significado aos bytes
    - Bytes podem representar caracteres, dados crus, imagens gráficas, voz, vídeo ou qualquer outra informação que uma aplicação pode precisar

Fluxos de Dados

- Operações de I/O frequentemente se tornam um gargalo
  - Esperam a entrada do disco ou do teclado
  - I/O de nível baixo
    - Não formatado (inconveniente para as pessoas)
    - Transferência byte-a-byte
    - Transferência em alta velocidade e em grande quantidades
  - I/O de nível alto
    - Formatado
    - Bytes agrupados (em inteiros, caracteres, strings etc.)
    - Bom para a maior parte das necessidade de I/O

Fluxos de Dados Clássico X Fluxo de Dados Padrão

- Bibliotecas de fluxos clássicos
  - Entrada/saída de **chars** (um byte)
    - Número limitado de caracteres (ASCII)
- Bibliotecas de fluxos de dados padrão
  - Algumas línguas precisam de alfabetos especiais
    - Além dos possíveis com **chars**
  - Definem tipos adicionais de caracteres que suportam caracteres Unicode
    - Tipo de caractere **wchar_t**
  - Pode fazer I/O com caracteres Unicode

Arquivos de Cabeçalhos da Biblioteca iostream

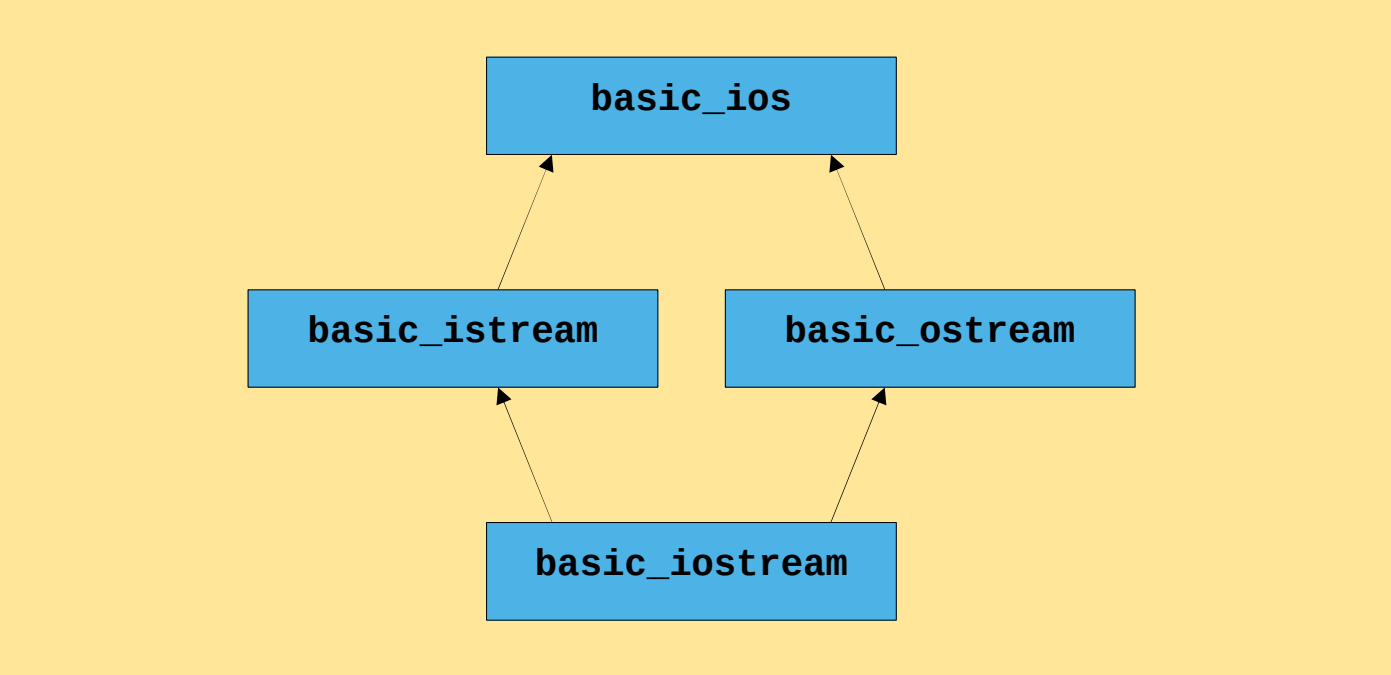
- Biblioteca **iostream**
  - Possui arquivos de cabeçalho com centenas de capacidades de I/O
  - **<iostream>**
    - Entrada padrão (standard input - **cin**)
    - Saída padrão (**cout**)
    - Erro não armazenado (**cerr**)
    - Erro armazenado (**clog**)
  - **<iomanip>**
    - Formata I/O com manipuladores de fluxos parametrizados
  - **<fstream>**
    - Operações de processamento de arquivo

Classes e Objetos para Entrada/Saída de Dados

- Biblioteca **iostream** tem classes templates para I/O
  - **basic_istream** (fluxo de entrada)
  - **basic_ostream** (fluxo de saída)
  - **basic_iostream** (fluxo de entrada e saída)
- Cada template tem uma especialização pré-definida
  - Permite **char** I/O
- **typedef** declara "alias" para as especializações
  - `typedef Card *CardPtr;`
    - **CardPtr** sinônimo para **Card ***
  - **typedefs istream**, **ostream**, **iostream**
    - Ex.: typedef **istream** representa uma especialização da basic_istream que permite entrada de **char**

Classes e Objetos para Entrada/Saída de Dados

- Templates "derivam" da **basic_ios**



Classes e Objetos para Entrada/Saída de Dados

- **<< e >>**
  - Operadores de inserção e extração de fluxos de dados
- **cin**
  - Objeto da classe **istream**
  - Conectado a entrada padrão (tipicamente o teclado)
  - **cin >> grade;**
    - Compilador determina o tipo de dados de grade
      - Chama o operador sobrecarregado apropriado
    - Não há informação extra de tipo necessária

Classes e Objetos para Entrada/Saída de Dados

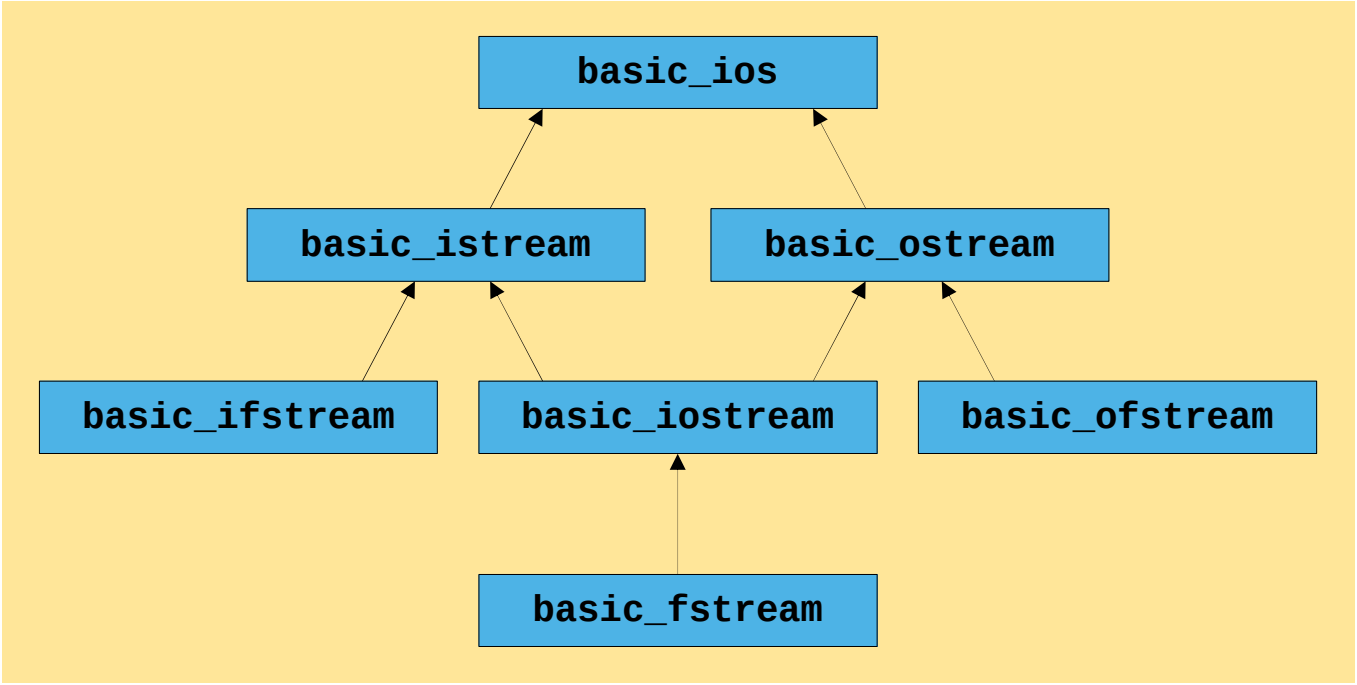
- **`cout`**
  - Objeto da classe **`ostream`**
  - Saída padrão (tipicamente tela do monitor)
  - **`cout << grade;`**
    - Como com **`cin`**, não há necessidade de informação de tipo
- **`cerr`**, **`clog`**
  - Objetos da classe **`ostream`**
  - Conectado ao dispositivo padrão de erro
  - **`cerr`** imprime na tela imediatamente: **`cerr << grade;`**
  - **`clog`** armazena a saída em *buffer*: **`clog << grade;`**
    - Até o *buffer* ficar cheio ou até ele ser esvaziado
    - Vantagem de desempenho

Classes e Objetos para Entrada/Saída de Dados

- Processamento de arquivos em C++
  - Classes templates
    - **basic_ifstream** (entrada de arquivo)
    - **basic_ofstream** (saída de arquivo)
    - **basic_fstream** (arquivo de I/O)
  - Especializações para **char** I/O
    - **typedefs** aliases para especializações de **char**
      - **ifstream**
      - **ofstream**
      - **fstream**

Classes e Objetos para Entrada/Saída de Dados

- Hierarquia dos templates



Saída de Fluxos Dados

- Saída
  - Usa **`ostream`**
  - Tipos de dados padrão (<<) podem ser:
    - Formatados e não formatados
    - Caracteres (função **`put`**)
    - Inteiros (decimal, octal, hexadecimal)
    - Números de ponto flutuante
      - Várias precisões, pontos decimais forçados, notação científica
    - Justificado, dados adicionados
    - Controle de maiúsculo/minúsculo

Saída de Variáveis char *

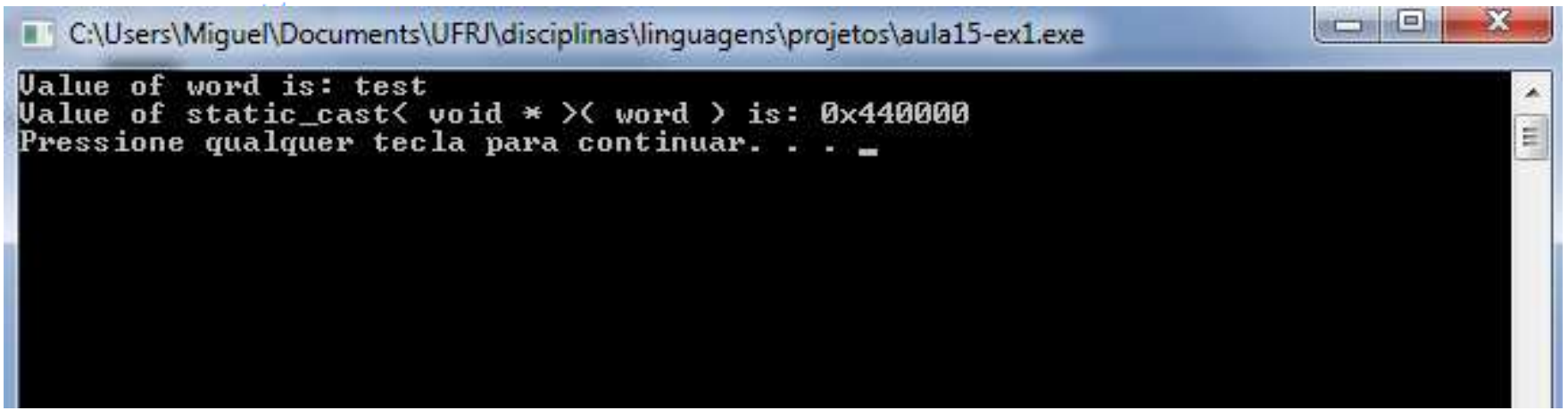
- C++ determina tipo de dados automaticamente
  - Geralmente uma melhoria (comparado ao C)
  - Tenta imprimir o valor de um **`char *`**
    - Endereço de memória do primeiro caractere
- Problema: Imprimir o endereço em memória do primeiro caractere de uma string
  - Operador **`<<`** é sobrecarregado para imprimir string terminada por **`NULL`**
  - Solução: cast para **`void *`**
    - Usado sempre para imprimir o valor de um ponteiro
    - Imprime o endereço em hexadecimal

Primeiro Exemplo Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 1
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>

using namespace std;

int main() {
        char *word = "test";

        // exibe valor do char *, então exibe valor do char *
        // static_cast para void *
        cout << "Value of word is: " << word << endl
             << "Value of static_cast< void * >( word ) is: "
             << static_cast< void * >( word ) << endl;

        return 0;
}
```

Primeiro Exemplo Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 1
 * Arquivo Principal
 * Autor: Miguel Campista
```




```
                << static_cast< void * >( word ) << endl;

        return 0;
}
```

Saída de Caractere Usando a Função Membro put

- Função **put**
  - Saída de caracteres
    - `cout.put( 'A' );`
  - Pode ser cascateado
    - `cout.put( 'A' ).put( '\n' );`
    - Operador ponto (`.`) avalia da esquerda para direita
  - Pode usar valor numérico (ASCII)
    - `cout.put( 65 )`
      - Imprime `'A'`

Entrada de Fluxo de Dados

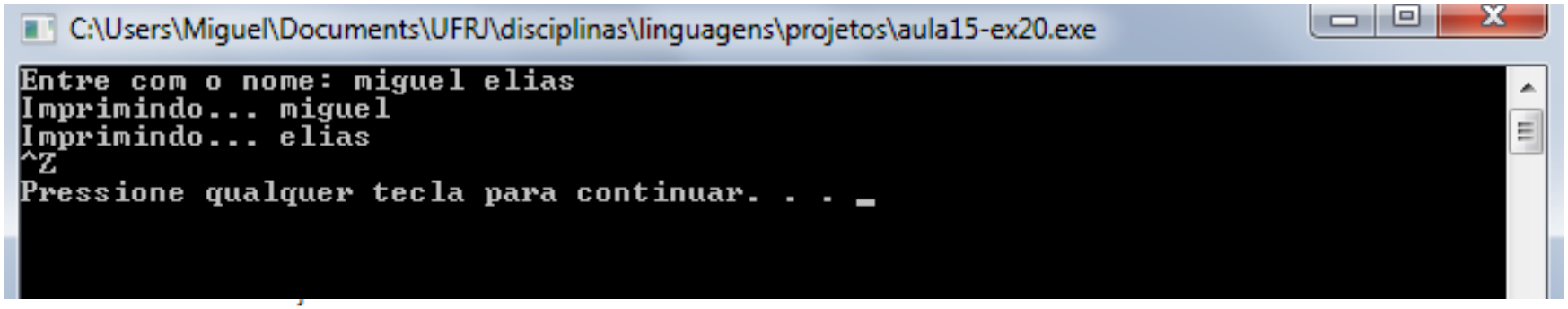
- Entrada formatada e não formatada
  - `Istream`
- Operador >>
  - Normalmente pula espaço em branco (brancos, tabs, nova linha)
    - Pode mudar isso
  - Retorna `0` quando EOF é encontrado
    - De outra maneira, retorna referência ao objeto **`cin`**
    - **`cin >> grade`**

Entrada de Fluxo de Dados

```
#include <iostream>

using namespace std;

int main() {
    char nome [100];

    cout << "Entre com o nome: ";

    while ((cin >> nome) != 0) {
          cout << "Imprimindo... " << nome << endl;
    }

    return 0;
}
```

Entrada de Fluxo de Dados

```
#include <iostream>

using namespace std;

int main() {
    char nome [100];
```



Funções Membro get e getline

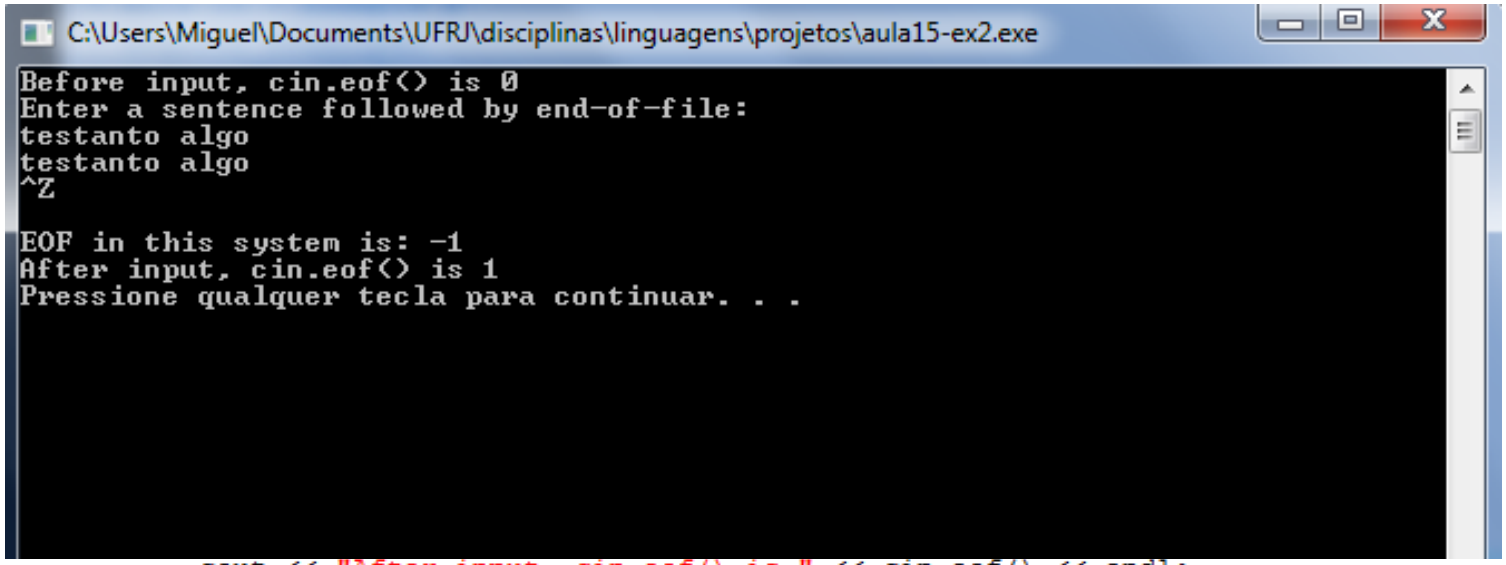
- Função **get**
  - **cin.get()**
  - Retorna um caractere do fluxo (mesmo espaço em branco)
    - Retorna **EOF** se o final do arquivo for encontrado
- End-of-file (EOF)
  - Indica final da entrada
    - *ctrl-z* em IBM-PCs
    - *ctrl-d* em UNIX e Macs
  - **cin.eof()**
    - Retorna **1** (**true**) se EOF ocorrer

Segundo Exemplo Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 2
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>

using namespace std;

int main() {
     int character; // usa int, porque char não pode representar EOF

     // solicita o usuário a entrar com linha de texto
     cout << "Before input, cin.eof() is " << cin.eof() << endl
          << "Enter a sentence followed by end-of-file:" << endl;

     // usa get para ler cada caractere; usa put para exibi-los
     while ( ( character = cin.get() ) != EOF )
           cout.put( character );

     // exibe caractere de final de arquivo
     cout << "\nEOF in this system is: " << character << endl;
     cout << "After input, cin.eof() is " << cin.eof() << endl;

     return 0;
}
```

Segundo Exemplo Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 2
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
```




```
    cout << "After input, cin.eof() is " << cin.eof() << endl;

    return 0;
}
```

Funções Membro get e getline

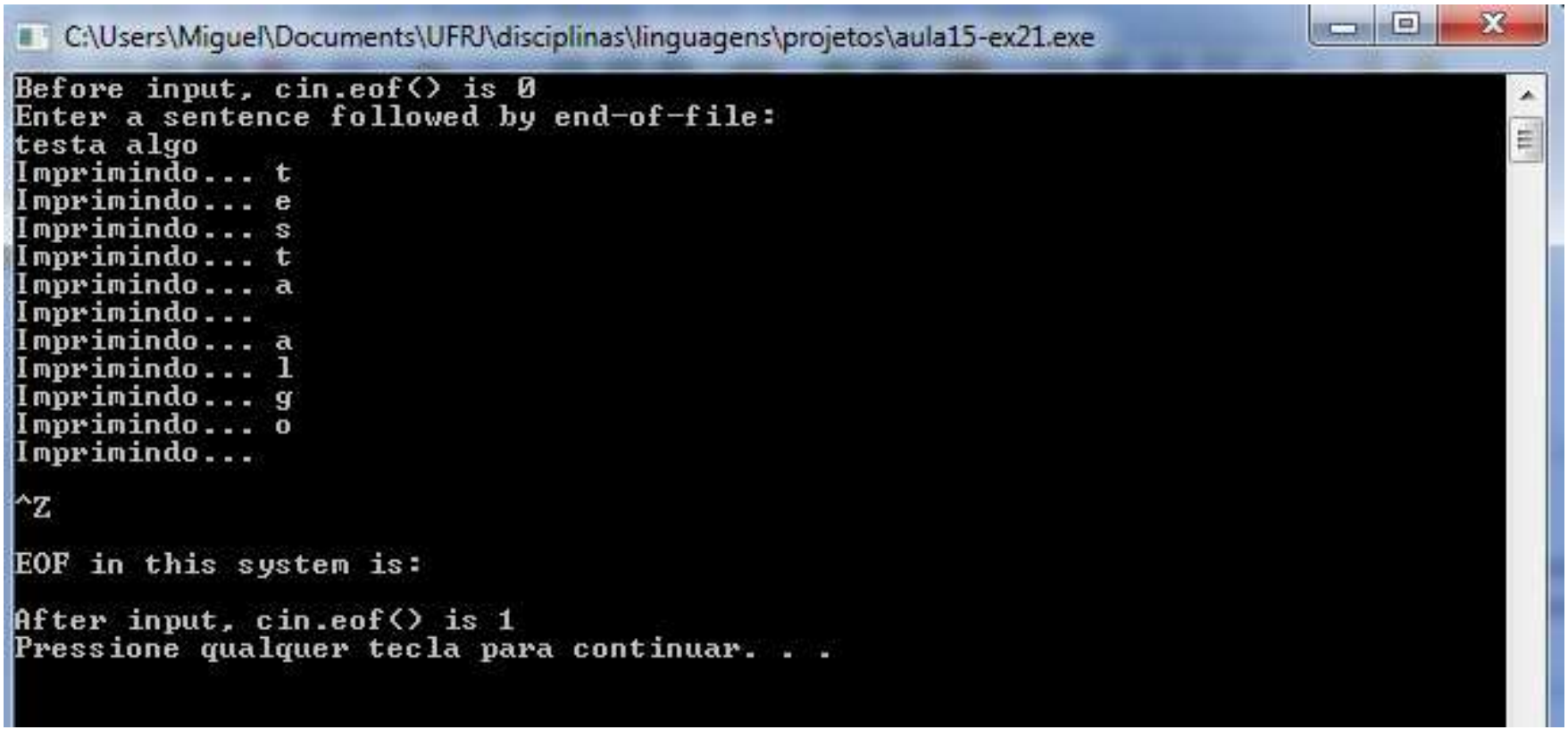
- **get(charRef)**
  - Com referência a um caractere como argumento
  - Recebe um caractere, armazena em **charRef**
    - Retorna referência para **istream**
- **get(charArray, size, delimiter)**
  - Lê até **size-1** caracteres lidos ou delimitador encontrado
    - Delimitador padrão **'\n'**
    - Delimitador fica em fluxo de dados de entrada
      - Pode remover com **cin.get()** ou **cin.ignore()**
  - Faz o array terminado por **NULL**

Terceiro Exemplo Usando E/S de Fluxo de Dados

```
#include <iostream>

using namespace std;

int main() {
     char character; // usa char porque a fç get aceita char como argumento

     // solicita o usuário a entrar com linha de texto
     cout << "Before input, cin.eof() is " << cin.eof() << endl
          << "Enter a sentence followed by end-of-file:" << endl;

     // usa get para ler cada caractere; usa put para exibi-los
     while ( cin.get (character) != 0 ) {
           cout << "Imprimindo... ";
           cout.put( character );
           cout << endl;
     }

     // exibe caractere de final de arquivo
     cout << "\nEOF in this system is: " << character << endl;
     cout << "After input, cin.eof() is " << cin.eof() << endl;

     return 0;
}
```

Terceiro Exemplo Usando E/S de Fluxo de Dados

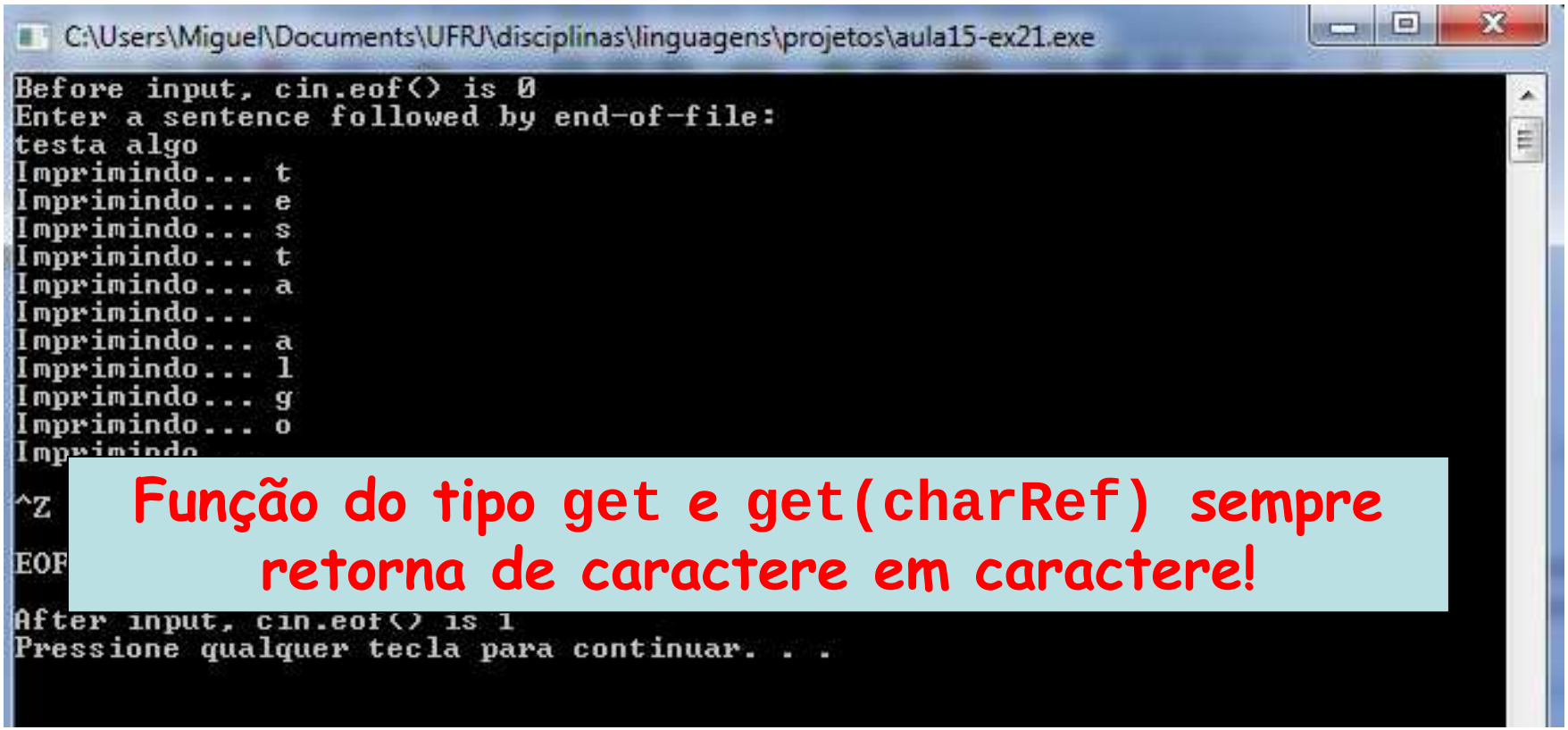
```
#include <iostream>
```




```
    return 0;
}
```

Terceiro Exemplo Usando E/S de Fluxo de Dados

```
#include <iostream>
```



Função do tipo get e get(charRef) sempre retorna de caractere em caractere!

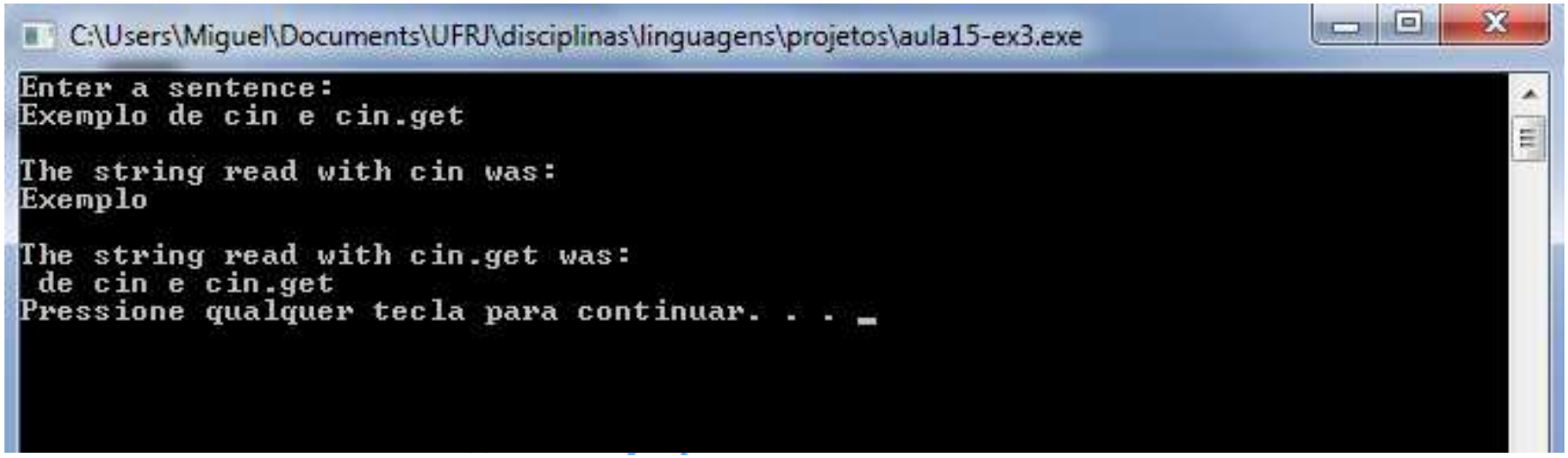
```
    return 0;
}
```

Terceiro Exemplo Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 3
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>

using namespace std;

int main() {
        // cria dois arrays de char, cada um com 80 elementos
        const int SIZE = 80;
        char buffer1[ SIZE ];
        char buffer2[ SIZE ];

        // usa cin para inserir caracteres no buffer1
        cout << "Enter a sentence:" << endl;
        cin >> buffer1;

        // exibe o conteúdo do buffer1
        cout << "\nThe string read with cin was:" << endl
             << buffer1 << endl << endl;

        // usa cin.get para inserir caracteres no buffer2
        cin.get( buffer2, SIZE );

        // exibe conteúdo do buffer2
        cout << "The string read with cin.get was:" << endl
             << buffer2 << endl;

        return 0;
}
```

Terceiro Exemplo Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 3
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>

using namespace std;
```




```
        cin.get( buffer2, SIZE );

        // exibe conteúdo do buffer2
        cout << "The string read with cin.get was:" << endl
             << buffer2 << endl;

        return 0;
}
```

Funções Membro get e getline

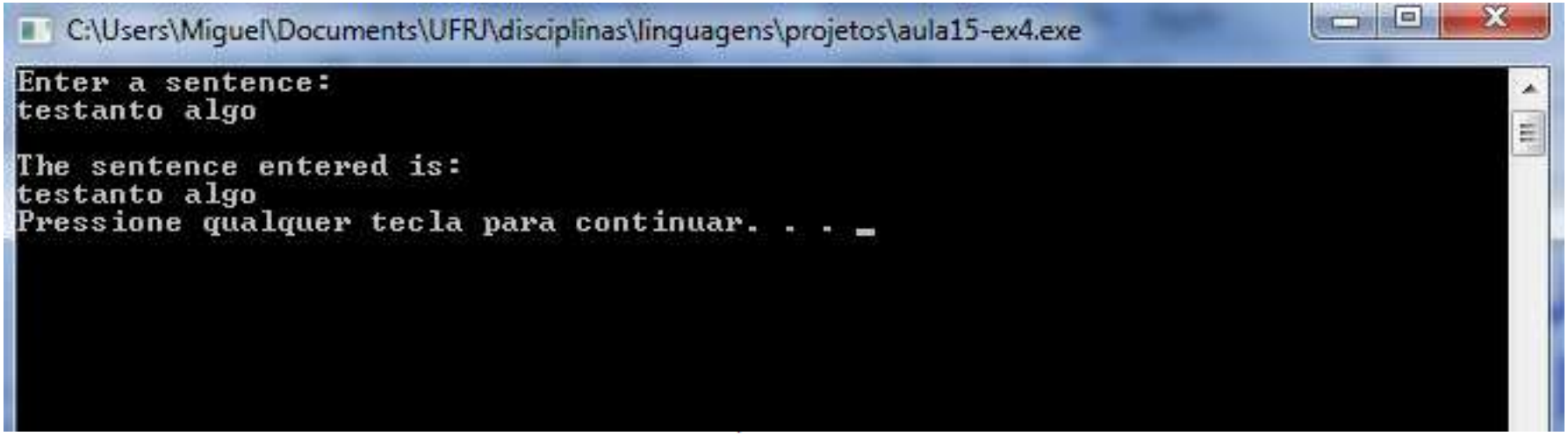
- **`getline(array, size, delimiter)`**
  - Como última versão do **`get`**
  - Lê **`size-1`** caracteres ou até delimitador for encontrado
    - Padrão: **`'\n'`**
  - Coloca caractere **`NULL`** no final do array
  - Diferença:
    - Remove delimitador do fluxo de dados de entrada e não deixa para a próxima execução da função
      - Semelhante ao exemplo anterior com o uso do **`get`**

Quarto Exemplo Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 4
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>

using namespace std;

int main() {
        const int SIZE = 80;
        char buffer[ SIZE ]; // cria array de char com 80 elementos

        // usa getline para inserir caracteres no buffer
        cout << "Enter a sentence:" << endl;
        cin.getline( buffer, SIZE );

        // exibe o conteúdo do buffer
        cout << "\nThe sentence entered is:" << endl
             << buffer << endl;

        return 0;
}
```

Quarto Exemplo Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 4
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
```




```
        return 0;
}
```

Funções Membro peek, putback e ignore de istream

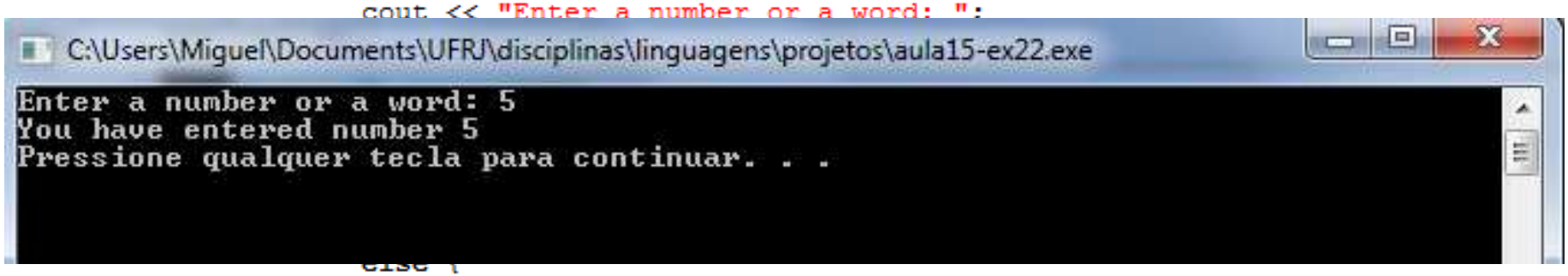
- **`ignore()`**
  - Descarta caracteres do fluxo (padrão 1 caractere)
  - Pára de descartar quando o delimitador é encontrado
    - Delimitador padrão é o **`EOF`**
- **`putback()`**
  - Coloca o caractere obtido pelo **`get()`** de volta no fluxo
- **`peek()`**
  - Retorna o próximo caractere no fluxo, mas não o remove

Funções Membro peek, putback e ignore de istream

```
#include <iostream>

using namespace std;

int main () {
    char c;
    int n;
    char str[256];
    cout << "Enter a number or a word: ";
    c = cin.get();
    if ( (c >= '0') && (c <= '9') ) {
        cin.putback(c);
        cin >> n;
        cout << "You have entered number " << n << endl;
     }
    else {
        cin.putback(c);
        cin >> str;
        cout << " You have entered word " << str << endl;
    }

    return 0;
}
```

Funções Membro peek, putback e ignore de istream

```
#include <iostream>

using namespace std;

int main () {
    char c;
    int n;
    char str[256];
    cout << "Enter a number or a word: ";
    cin.putback(c);
    cin >> str;
    cout << " You have entered word " << str << endl;
}

    return 0;
}
```

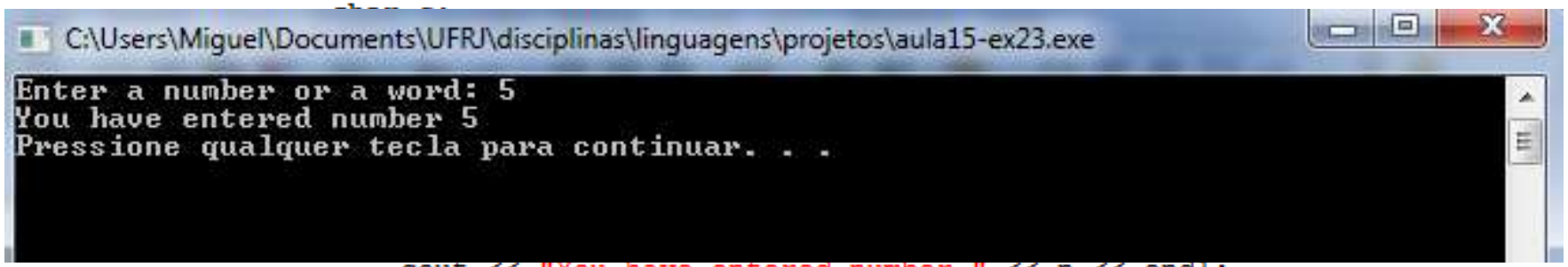


Funções Membro peek, putback e ignore de istream

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main () {
    char c;
    int n;
    char str[256];
    cout << "Enter a number or a word: ";
    c = cin.peek();
    if ( (c >= '0') && (c <= '9') ) {
        cin >> n;
        cout << "You have entered number " << n << endl;
     }
    else {
        cin >> str;
        cout << " You have entered word " << str << endl;
    }

    return 0;
}
```

Funções Membro peek, putback e ignore de istream

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main () {
```




```
            cout << "You have entered number " << n << endl;
        }
        else {
            cin >> str;
            cout << " You have entered word " << str << endl;
        }

        return 0;
}
```

Tipo Seguro I/O

- **<< e >>**
  - Sobrecarregado para aceitar tipos específicos de dados
- Se dado inesperado, então é processado
  - Bits de erro são ligados
  - Usuário pode testar bits para ver se I/O falhou

I/O Não Formatado Usando `read`, `write` e `gcount`

- I/O não formatado
  - **`read`** (membro de **`istream`**)
    - Insere bytes sem formatação no array de caracteres
    - Se caracteres lidos não forem suficientes, **`failbit`** é ligado
    - **`gcount()`** retorna o número de caracteres lidos na última operação

I/O Não Formatado Usando read, write e gcount

- I/O não formatado
  - **write (ostream)**
    - Exibe bytes do array de caractere
      - Pára quando caractere null é achado

```
char buffer[] = "HAPPY BIRTHDAY";

cout.write( buffer, 10 );
```

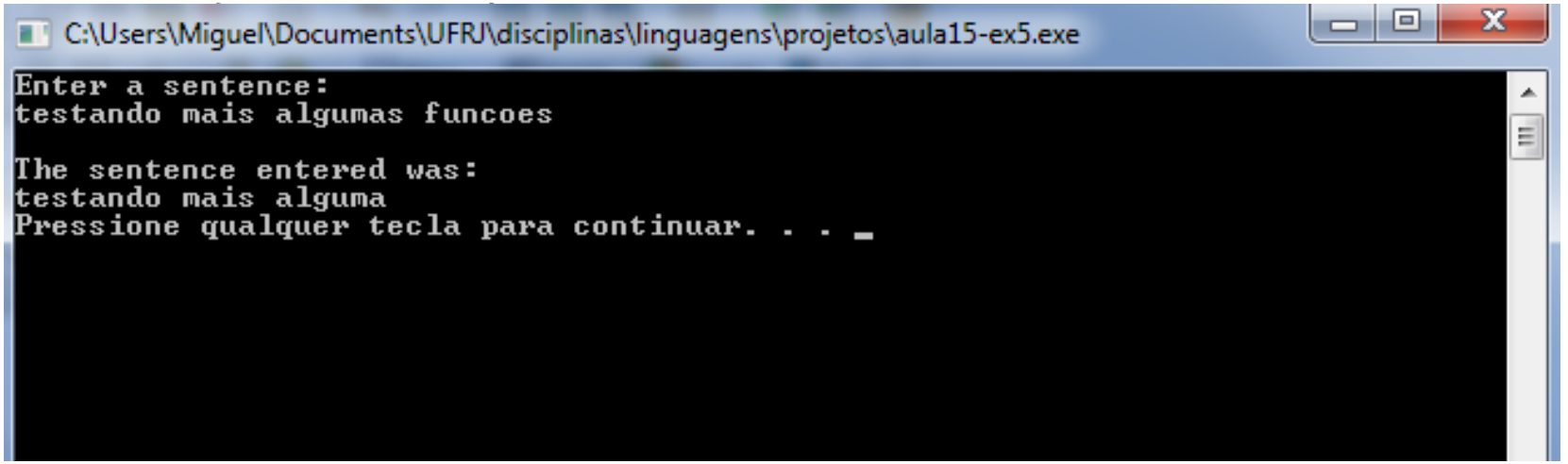
      - Exibe apenas os primeiros 10 caracteres

Quinto Exemplo Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 5
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>

using namespace std;

int main() {
        const int SIZE = 80;
        char buffer[ SIZE ]; // cria array de char com 80 elementos

        // usa getline para inserir caracteres no buffer
        cout << "Enter a sentence:" << endl;
        cin.read( buffer, 20 );

        // exibe o conteúdo do buffer
        cout << "\nThe sentence entered was:" << endl;
        cout.write( buffer, cin.gcount() );
        cout << endl;

        return 0;
}
```

Quinto Exemplo Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 5
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
```




```
        cout << endl;

        return 0;
}
```

Introdução aos Manipuladores de Fluxos

- Manipuladores de fluxo desempenham tarefas de formatação
  - Largura dos campos
  - Precisão
  - Indicadores de formatação
  - Preenchimento de caracteres em campos
  - Liberação (*flushing*) de fluxos
  - Inserção de nova linha (*newline*) no fluxo de saída
  - Desprezo de espaço em branco no fluxo de entrada

Base de Fluxo Integral: dec, oct, hex e setbase

- Inteiros normalmente em base 10 (decimal)
  - Manipuladores de fluxo para mudar a base
    - **hex** (base 16)
    - **oct** (base 8)
    - **dec** (reinicia para a base 10)
    - **cout << hex << myInteger**
  - **setbase(newBase)**
    - Uma das 8, 10 ou 16
  - Base é a mesma até que mudada explicitamente

Base de Fluxo Integral: dec, oct, hex e setbase

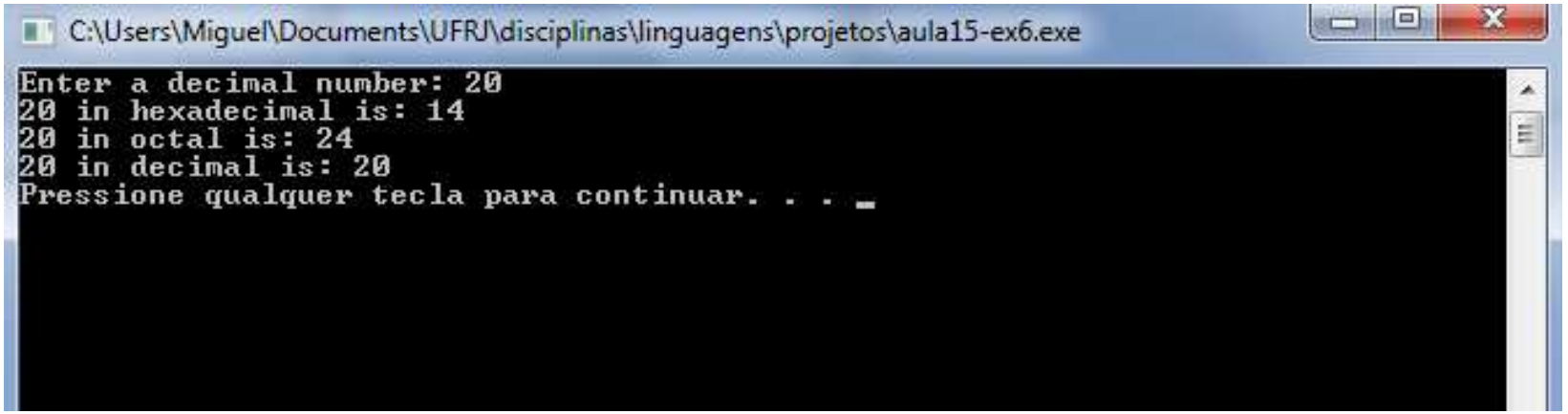
- Manipuladores de fluxos parametrizados
  - Usa cabeçalho **<iomanip>**
  - Recebe argumento (como **setbase**)

Sexto Exemplo Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 6
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
	int number;

	cout << "Enter a decimal number: ";
	cin >> number; // número de entrada

	// usa manipulador de fluxo hex para mostrar número hexadecimal
	cout << number << " in hexadecimal is: " << hex
	     << number << endl;
	// usa manipulador de fluxo oct para mostrar número octal
	cout << dec << number << " in octal is: "
	     << oct << number << endl;

	// usa manipulador de fluxo setbase para mostrar número decimal
	cout << setbase( 10 ) << number << " in decimal is: "
	     << number << endl;

	return 0;
}
```

Sexto Exemplo Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 6
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iomanip>
```



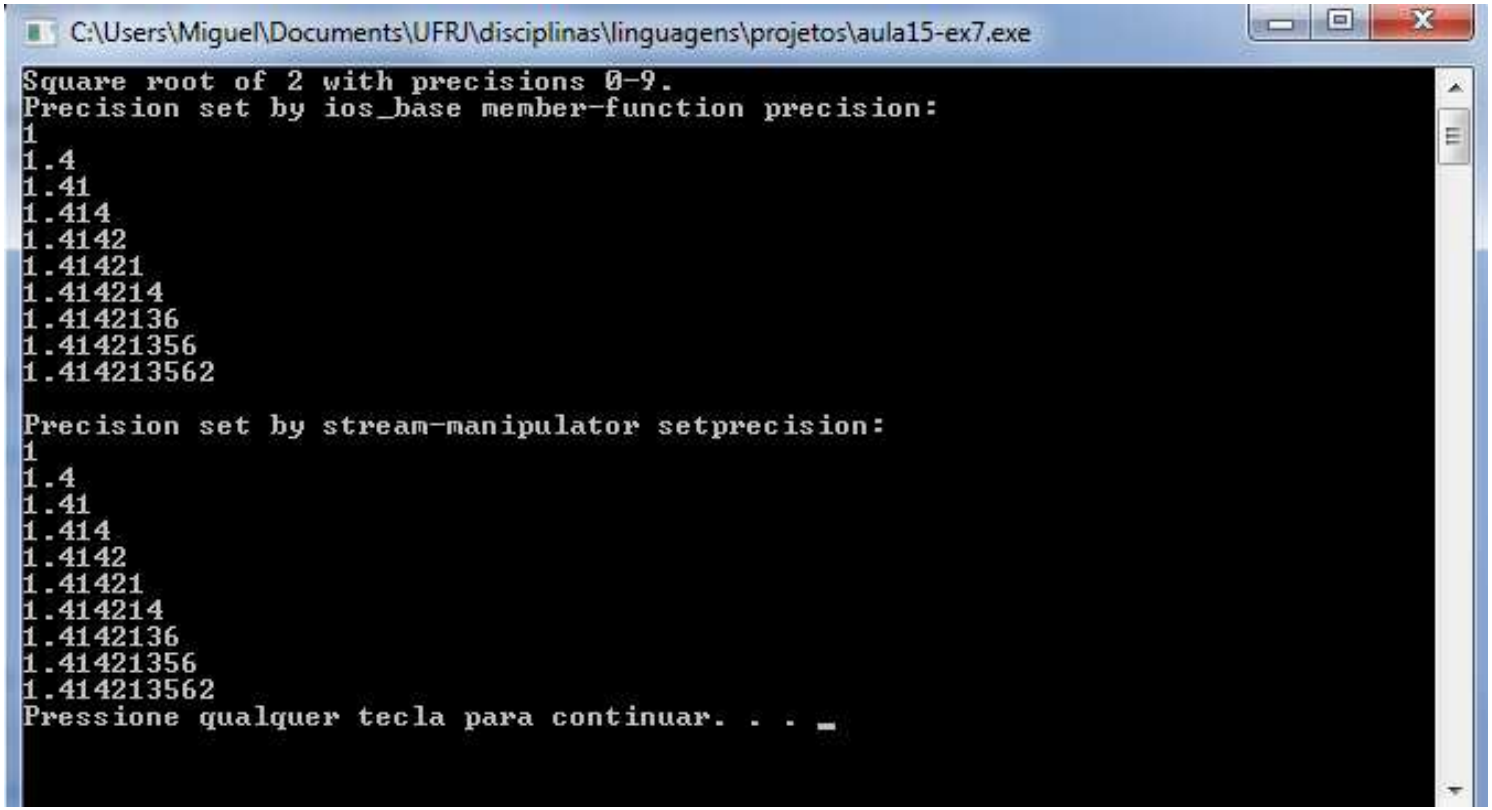

```
        // usa manipulador de fluxo oct para mostrar número octal
        cout << dec << number << " in octal is: "
             << oct << number << endl;

        // usa manipulador de fluxo setbase para mostrar número decimal
        cout << setbase( 10 ) << number << " in decimal is: "
             << number << endl;

        return 0;
}
```

Precisão de Ponto Flutuante (precision, setprecision)

- Definir precisão dos números de ponto flutuante
  - Número de dígitos para a direita do decimal
  - Manipulador de precisão **setprecision**
    - Passa número de casas decimais
    - **cout << setprecision(5)**
  - Função membro **precision**
    - **cout.precision(newPrecision)**
    - Sem argumentos, retorna precisão atual
  - Definições permanecem até mudadas explicitamente

```cpp
/*
 * Aula 15 - Exemplo 7
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iomanip>
#include <cmath>

using namespace std;

int main() {
	double root2 = sqrt( 2.0 ); // calcula raiz quadrada de 2
	int places;

	cout << "Square root of 2 with precisions 0-9." << endl
		<< "Precision set by ios_base member-function "
		<< "precision:" << endl;

	cout << fixed; // usa precisão fixa

	// exibe raiz quadrada usando função de precisão da ios_base
	for ( places = 0; places <= 9; places++ ) {
		cout.precision( places );
		cout << root2 << endl;
	}

	cout << "\nPrecision set by stream-manipulator "
		<< "setprecision:" << endl;

	// define precisão para cada dígito, então exibe raiz quadrada
	for ( places = 0; places <= 9; places++ )
		cout << setprecision( places ) << root2 << endl;

	return 0;
}
```

```
/*
 * Aula 15 - Exemplo 7
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iomanip>
#include <cmath>
```




```
        // define precisão para cada dígito, então exibe raiz quadrada
        for ( places = 0; places <= 9; places++ )
             cout << setprecision( places ) << root2 << endl;

        return 0;
}
```

Largura do Campo (`width`, `setw`)

- Função membro **`width`** (classe base **`ios_base`**)
  - **`cin.width(5)`**
  - Define largura do campo
    - Número de posições de caracteres para a saída
    - Número máximo de caracteres que devem ser recebidos
  - Retorna largura anterior
  - Preenche caracteres/enchimento
    - Usada quando a saída é muito pequena em largura
    - Saídas grandes são impressas (não são truncadas)
  - Pode também usar manipulador de fluxo **`setw`**
  - Não é uma configuração persistente

Largura do Campo (`width`, `setw`)

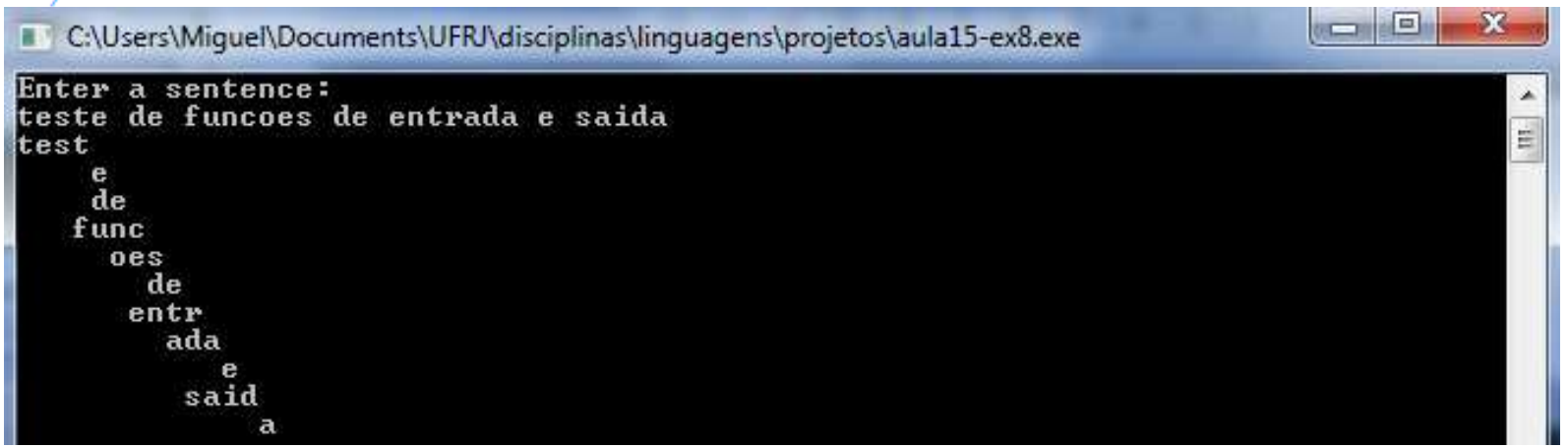
- Quando leitura para arrays de **`char`**
  - Lê menos um (1) caractere, pois deixa espaço para **`NULL`**

Oitavo Exemplo Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 8
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>

using namespace std;

int main () {
	int widthValue = 4;
	char sentence[ 10 ];

	cout << "Enter a sentence:" << endl;
	cin.width( 5 ); // insere somente 5 caracteres das sentenças

	// define campo largura, então exibe caracteres baseado nessa largura
	while ( cin >> sentence ) {
		cout.width( widthValue++ );
		cout << sentence << endl;
		cin.width( 5 ); // insere mais 5 caracteres da sentença
	}

	return 0;
}
```

Oitavo Exemplo Usando E/S de Fluxo de Dados

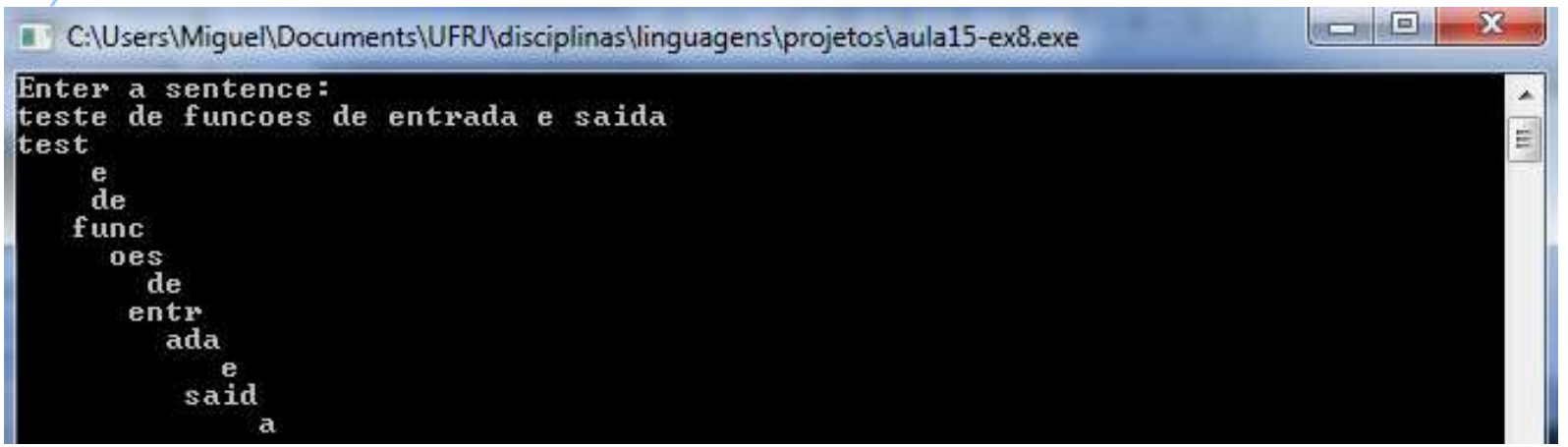
```
/*
```




```
        cout << "Enter a sentence:" << endl;
        cin.width( 5 ); // insere somente 5 caracteres das sentenças

        // define campo largura, então exibe caracteres baseado nessa largura
        while ( cin >> sentence ) {
              cout.width( widthValue++ );
              cout << sentence << endl;
              cin.width( 5 ); // insere mais 5 caracteres da sentença
        }

        return 0;
}
```

Oitavo Exemplo Usando E/S de Fluxo de Dados

```
/*
```




```
cout << "Enter a sentence:" << endl;
cin.width( 5 ); // insere somente 5 caracteres das sentenças

// define campo largura, então exibe caracteres baseado nessa largura
while ( cin >> sentence ) {
      cout.width( widthValue++ );
      cout << sentence << endl;
      cin.width( 5 ); // insere mais 5 caracteres da sentença
```

Pára de imprimir principalmente nos espaços em branco ou após os (cinco – um) caracteres permitidos para leitura.

Manipuladores Definidos por Usuários

- Manipuladores definidos por usuários
  - Não parametrizado
    - Ex.:

```
ostream& bell( ostream& output )
{
 return output << '\a';  // toca beep
}
```

  - `\a` - campainha
  - `\r` - carriage return
  - `\t` - tab

Nono Exemplo Usando E/S de Fluxo de Dados

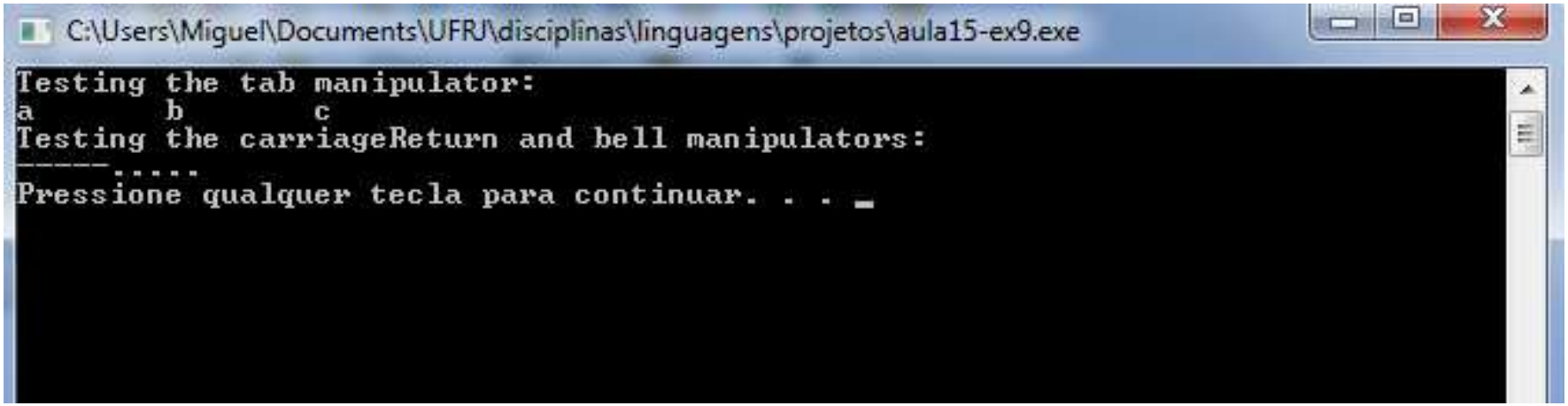
```
/*
 * Aula 15 - Exemplo 9
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>

using namespace std;

// manipulador de campainha (usando sequência de escape \a)
ostream& bell( ostream& output ) {
      return output << '\a';  // toca beep do sistema
}

// manipulador de carriageReturn manipulator (usando sequência de escape \r)
ostream& carriageReturn( ostream& output ) {
      return output << '\r';  // executa o carriage return
}

// manipulador tab (usando sequeência de escape \t)
ostream& tab( ostream& output ) {
      return output << '\t';  // executa tab
}

// manipulador endLine manipulator (usando sequência escape \n e função
// membro flush
ostream& endLine( ostream& output ) {
      return output << '\n' << flush;  // executa end of line
}
```

Nono Exemplo Usando E/S de Fluxo de Dados

```
int main() {
      // usa manipuladores tab e endLine
      cout << "Testing the tab manipulator:" << endLine
           << 'a' << tab << 'b' << tab << 'c' << endLine;

      cout << "Testing the carriageReturn and bell manipulators:"
           << endLine << ".........";

      cout << bell; // usa manipulador campainha

      // usa manipuladores carriageReturn e endLine
      cout << carriageReturn << "-----" << endLine;

      return 0;
}
```

Nono Exemplo Usando E/S de Fluxo de Dados

```
int main() {
      // usa manipuladores tab e endLine
      cout << "Testing the tab manipulator:" << endLine
           << 'a' << tab << 'b' << tab << 'c' << endLine;
```



Estados de Formato de Fluxos e Manipuladores de Fluxos

- Muitos manipuladores de formatação de fluxo
  - Herdaram da classe **`ios_base`**

Zeros à Direita e Pontos Decimais (showpoint)

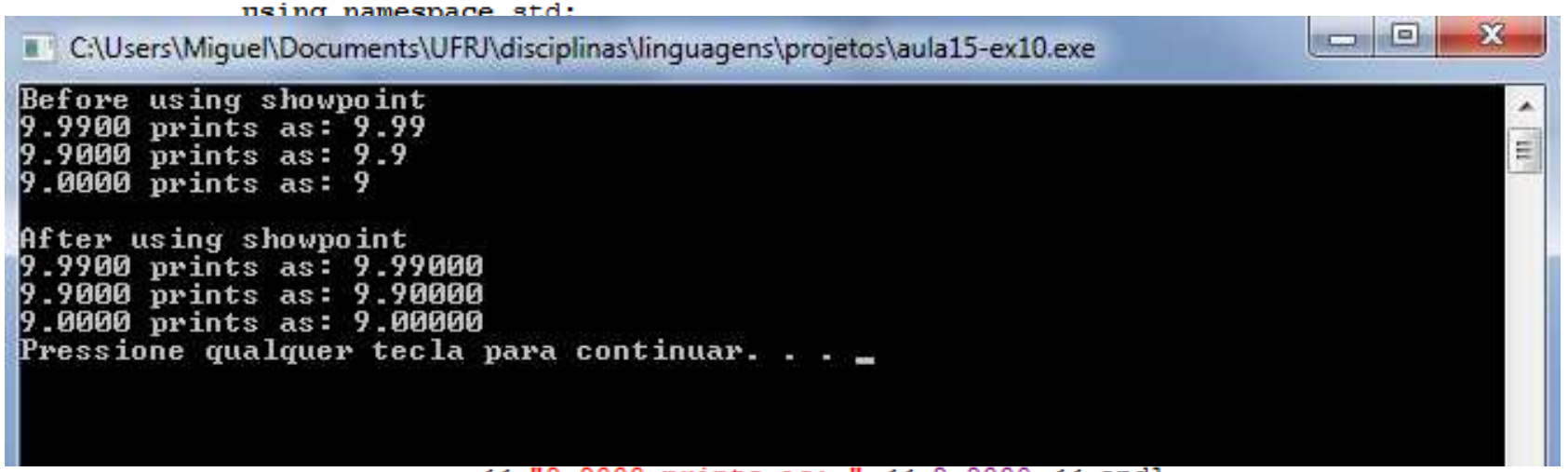
- **showpoint**
  - Força números decimais a imprimir com zeros à direita
  - Para o número decimal 79.0
    - 79 sem **showpoint**
    - 79.000000 com **showpoint** (até o nível de precisão)
  - Redefine com **noshowpoint**

Décimo Exemplo Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 10
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>

using namespace std;

int main() {
        // exibe valores double com formatação de fluxo padrão
        cout << "Before using showpoint" << endl
             << "9.9900 prints as: " << 9.9900 << endl
             << "9.9000 prints as: " << 9.9000 << endl
             << "9.0000 prints as: " << 9.0000 << endl << endl;

        // display double value after showpoint
        cout << showpoint
             << "After using showpoint" << endl
             << "9.9900 prints as: " << 9.9900 << endl
             << "9.9000 prints as: " << 9.9000 << endl
             << "9.0000 prints as: " << 9.0000 << endl;

        return 0;
}
```

Décimo Exemplo Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 10
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>

using namespace std;
```




```
                << "9.9000 prints as: " << 9.9000 << endl
                << "9.0000 prints as: " << 9.0000 << endl;

        return 0;
}
```

Justificação (left, right e internal)

- Manipulador de fluxo **left**
  - Justificado à esquerda, preenchimento à direita
  - Manipulador de fluxo **right**
    - Justificado à direita, preenchimento à direita

- **internal**
  - Sinal do número justificado à esquerda
  - Valor do número justificado à direita

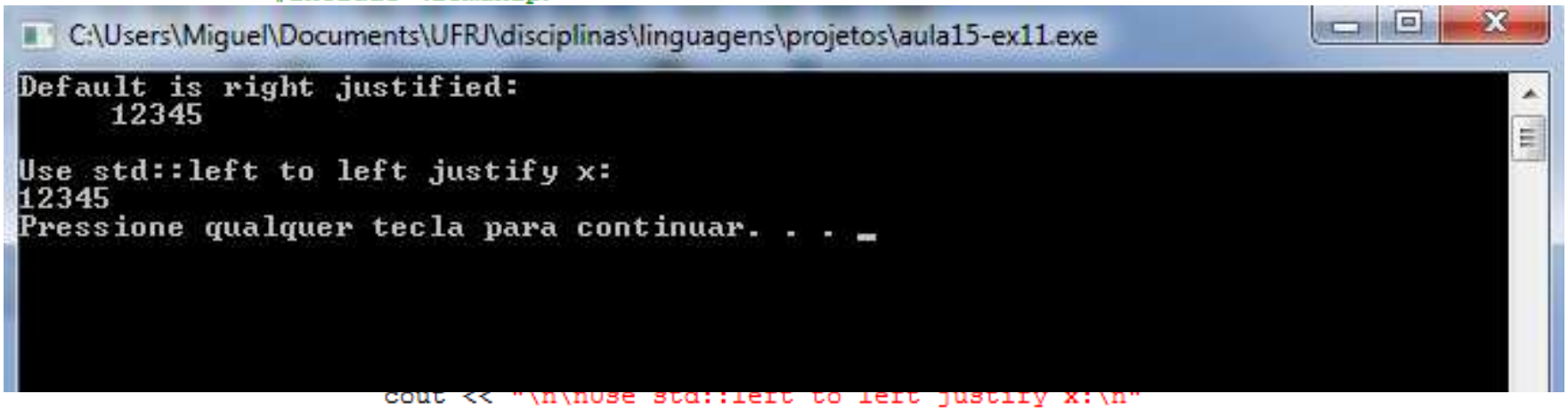
    + 123
  - **showpos** força impressão do sinal (positivo ou negativo à esquerda)
    - Remove com **noshowpos**

Décimo Primeiro Exemplo
Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 11
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
	int x = 12345;

	// exibe x justicado à direita (padrão)
	cout << "Default is right justified:" << endl
		<< setw( 10 ) << x;

	// usa manipulador left para exibir x justificado à esquerda
	cout << "\n\nUse std::left to left justify x:\n"
		<< left << setw( 10 ) << x << endl;

	return 0;
}
```

Décimo Primeiro Exemplo
Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 11
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iomanip>
```



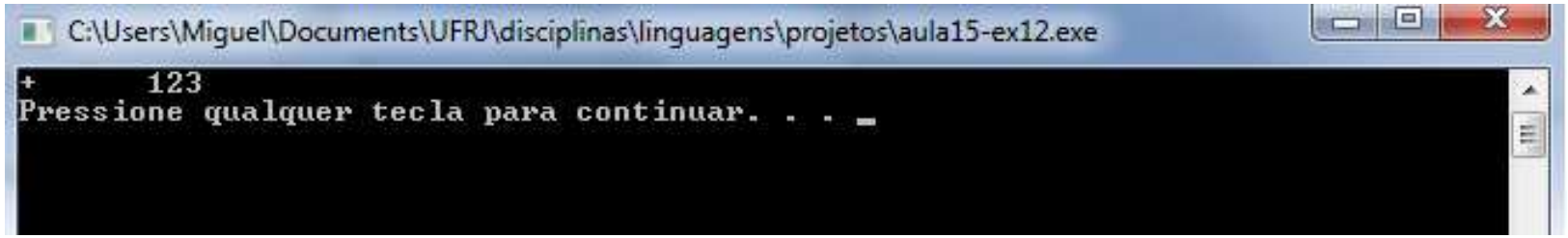

```
        cout << "\n\nUse std::left to left justify x:\n"
             << left << setw( 10 ) << x << endl;

        return 0;
}
```

Décimo Segundo Exemplo
Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 12
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
	// exibe valor com espaçamento interno e mais o sinal
	cout << internal << showpos << setw( 10 ) << 123 << endl;

	return 0;
}
```

Décimo Segundo Exemplo
Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 12
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iomanip>
```




```
        return 0;
}
```

Preenchimento (fill, setfill)

- Preenchimento
  - Função membro **fill**
    - **cout.fill('*')**
  - Manipulador de fluxo **setfill**
    - **setfill('^')**

Décimo Terceiro Exemplo Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 13
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
	int x = 10000;

	// exibe x
	cout << x << " printed as int right and left justified\n"
	     << "and as hex with internal justification.\n"
	     << "Using the default pad character (space):" << endl;

	// exibe x com sinal de mais
	cout << showbase << setw( 10 ) << x << endl;

	// exibe x justificado à esquerda
	cout << left << setw( 10 ) << x << endl;

	// exibe x como hex com justificação interna
	cout << internal << setw( 10 ) << hex << x << endl << endl;

	cout << "Using various padding characters:" << endl;

	// exibe x using com caracteres de preenchimento (justificado à direita)
	cout << right;
	cout.fill( '*' );
	cout << setw( 10 ) << dec << x << endl;
```

Décimo Terceiro Exemplo
Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 13
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
        int x = 10000;

        // exibe x
        cout << x << " printed as int right and left justified\n"
             << "and as hex with internal justification.\n"
             << "Using the default pad character (space):" << endl;

        // exibe x com sinal de mais
        cout << showbase << setw( 10 ) << x << endl;

        // exibe x justificado à esquerda
        cout << left << setw( 10 ) << x << endl;

        // exibe x como hex com justificação interna
                                                          << endl;
                                                          ;

        // exibe x using com caracteres de preenchimento (justificado à direita)
        cout << right;
        cout.fill( '*' );
        cout << setw( 10 ) << dec << x << endl;
```

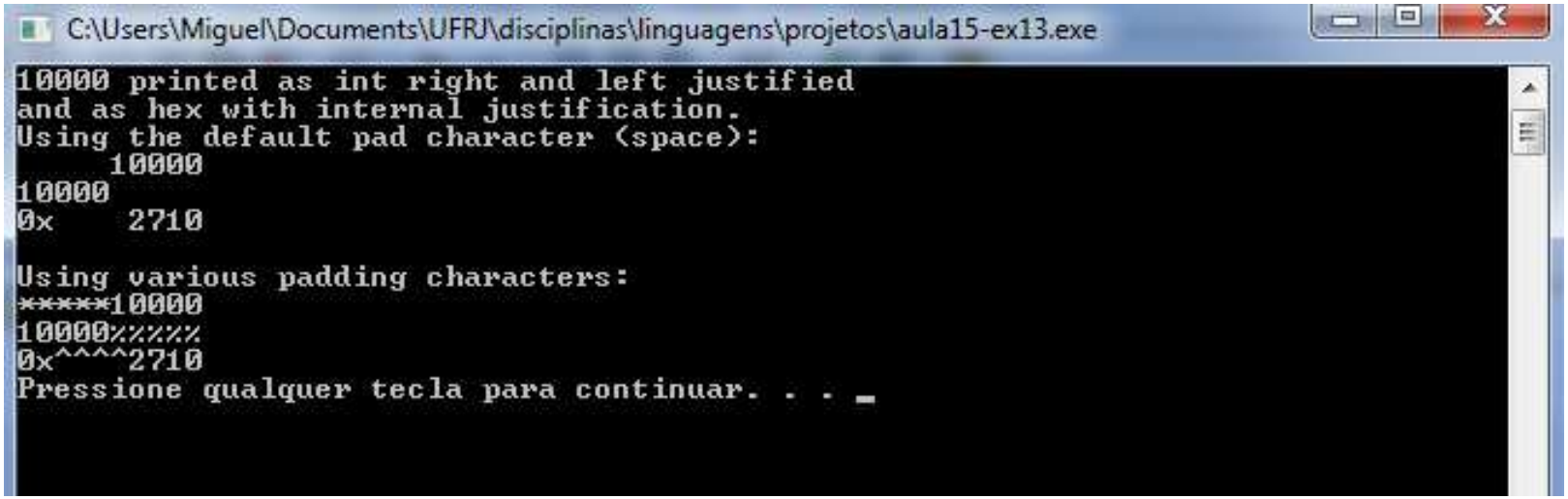
Força a exibição da base numérica

Décimo Terceiro Exemplo
Usando E/S de Fluxo de Dados

```
        // exibe x usando caracteres de preenchimento (justificados à esquerda)
        cout << left << setw( 10 ) << setfill( '%' ) << x << endl;

        // exibe x usando caracteres de preenchimento (justificados internamente)
        cout << internal << setw( 10 ) << setfill( '^' ) << hex
             << x << endl;

        return 0;
}
```

Décimo Terceiro Exemplo
Usando E/S de Fluxo de Dados

```
// exibe x usando caracteres de preenchimento (justificados à esquerda)
cout << left << setw( 10 ) << setfill( '%' ) << x << endl;
```



Base do Fluxo Integral (dec, oct, hex, showbase)

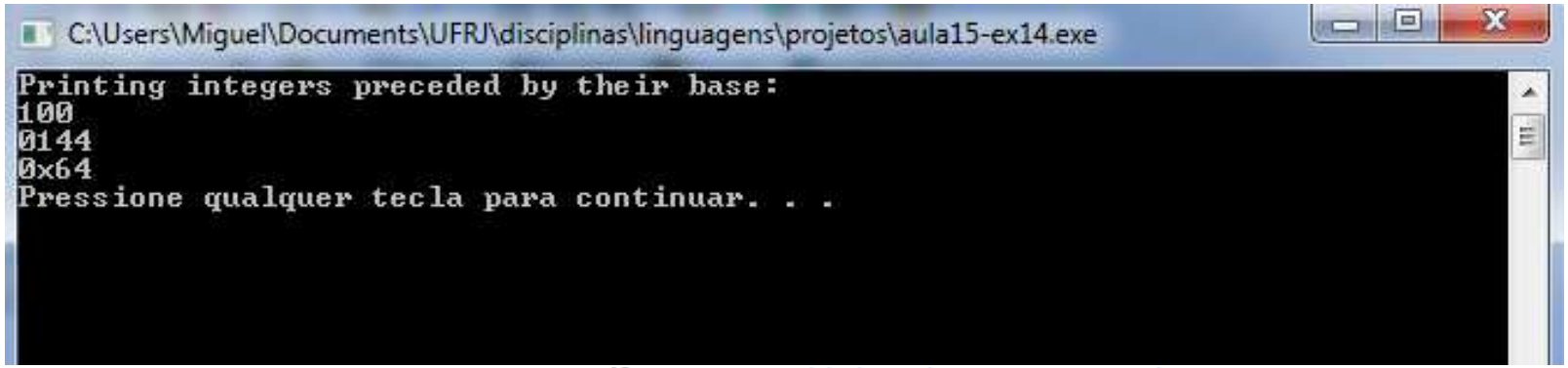
- Impressão de inteiro em várias bases
  - **dec**, **oct**, **hex**
- Extração de fluxo → Bases possíveis
  - Número decimal padrão
  - Precedido por **0** para octal
  - Precedido por **0x** ou **0X** para hex
- **showbase**
  - Força a base do número a ser mostrado
  - Remove com **noshowbase**

Décimo Quarto Exemplo Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 14
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
	int x = 100;

	// usa showbase para mostrar base do número
	cout << "Printing integers preceded by their base:" << endl
	     << showbase;

	cout << x << endl;         // imprime valor decimal
	cout << oct << x << endl; // imprime valor octal
	cout << hex << x << endl; // imprime valor hexadecimal

	return 0;
}
```

Décimo Quarto Exemplo Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 14
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iomanip>
```




```
        cout << x << endl;         // imprime valor decimal
        cout << oct << x << endl; // imprime valor octal
        cout << hex << x << endl; // imprime valor hexadecimal

        return 0;
}
```

Números de Ponto Flutuante, Notação Científica e Fixa

- Manipulador de fluxo **`scientific`**
  - Força notação científica
    - **`1.946000e+009`**

- Manipulador de fluxo **`fixed`**
  - Força formato fixo do ponto
  - Imprime número de decimais especificado para precisão
    - **`1946000000.000000`**

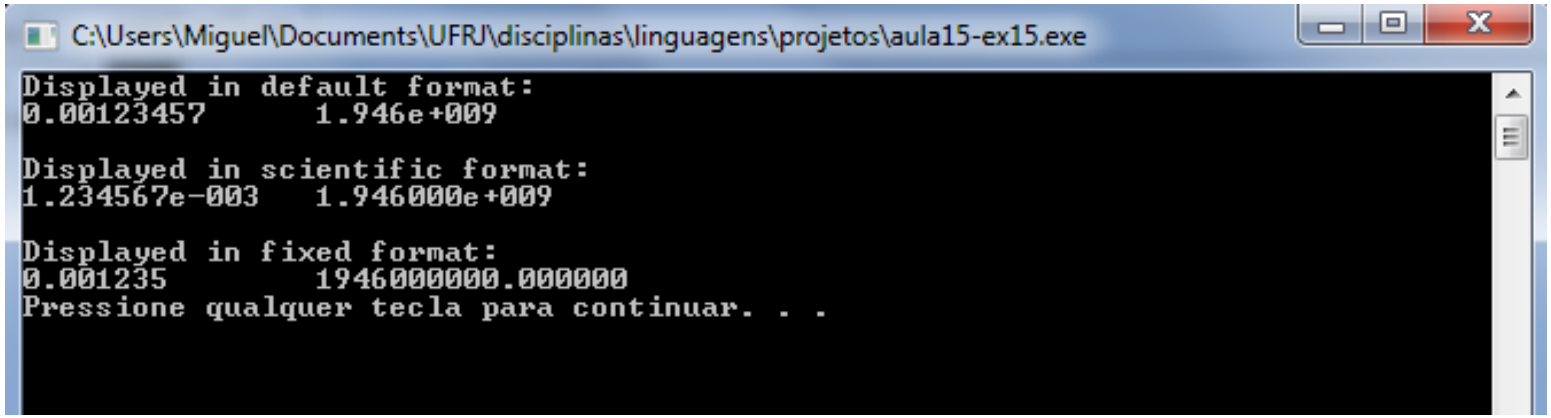
- Se nenhum manipulador for especificado
  - Formato do número determina como ele aparece

Décimo Quinto Exemplo Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 15
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
	double x = 0.001234567;
	double y = 1.946e9;

	// exibe x e y no formato padrão
	cout << "Displayed in default format:" << endl
	     << x << '\t' << y << endl;

	// exibe x e y no formato científico
	cout << "\nDisplayed in scientific format:" << endl
	     << scientific << x << '\t' << y << endl;

	// exibe x e y no formato fixo
	cout << "\nDisplayed in fixed format:" << endl
	     << fixed << x << '\t' << y << endl;

	return 0;
}
```

Décimo Quinto Exemplo Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 15
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iomanip>
```




```
        // exibe x e y no formato científico
        cout << "\nDisplayed in scientific format:" << endl
             << scientific << x << '\t' << y << endl;

        // exibe x e y no formato fixo
        cout << "\nDisplayed in fixed format:" << endl
             << fixed << x << '\t' << y << endl;

        return 0;
}
```

Controle de Maiúscula/Minúscula (uppercase)

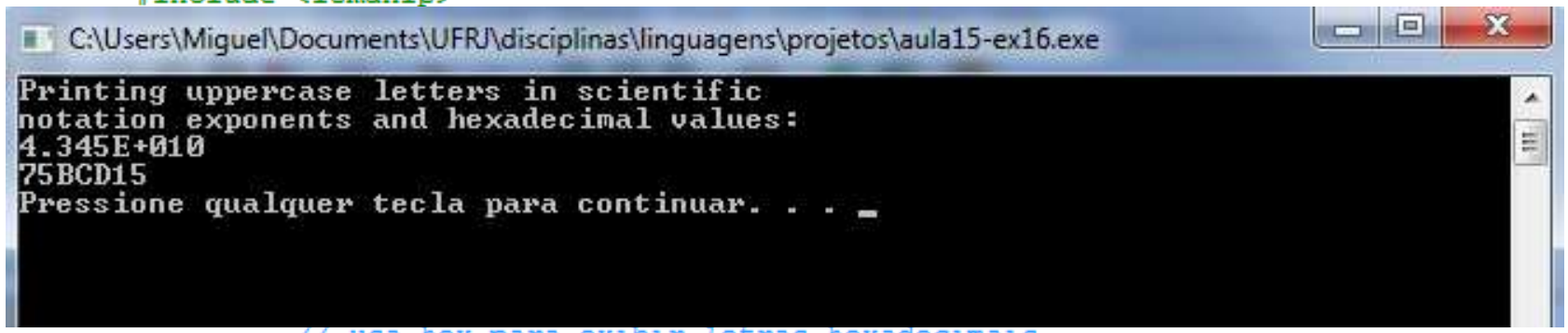
- Manipulador de fluxo **uppercase**
  - 'E' maiúsculo em notação científica
    - **1E10**
  - 'X' maiúsculo em notação hex e letras hex maiúsculas
    - **0XABCD**
  - Por padrão, letra minúscula
  - Reiniciar com **nouppercase**

Décimo Sexto Exemplo Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 16
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
        cout << "Printing uppercase letters in scientific" << endl
             << "notation exponents and hexadecimal values:" << endl;

        // usa uppercase para exibir letras maiúsculas;
        // usa hex para exibir letras hexadecimais
        cout << uppercase << 4.345e10 << endl << hex << 123456789
             << endl;

        return 0;
}
```

Décimo Sexto Exemplo Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 16
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iomanip>
```




```
        // usa hex para exibir letras hexadecimais
        cout << uppercase << 4.345e10 << endl << hex << 123456789
             << endl;

        return 0;
}
```

Especificação de Formato Booleano (boolalpha)

- Tipo de dados **bool**
  - Valores **true** ou **false**
  - Saída **0** (**false**) ou **1** (**true**) quando usada com **<<**
    - Exibido como inteiros

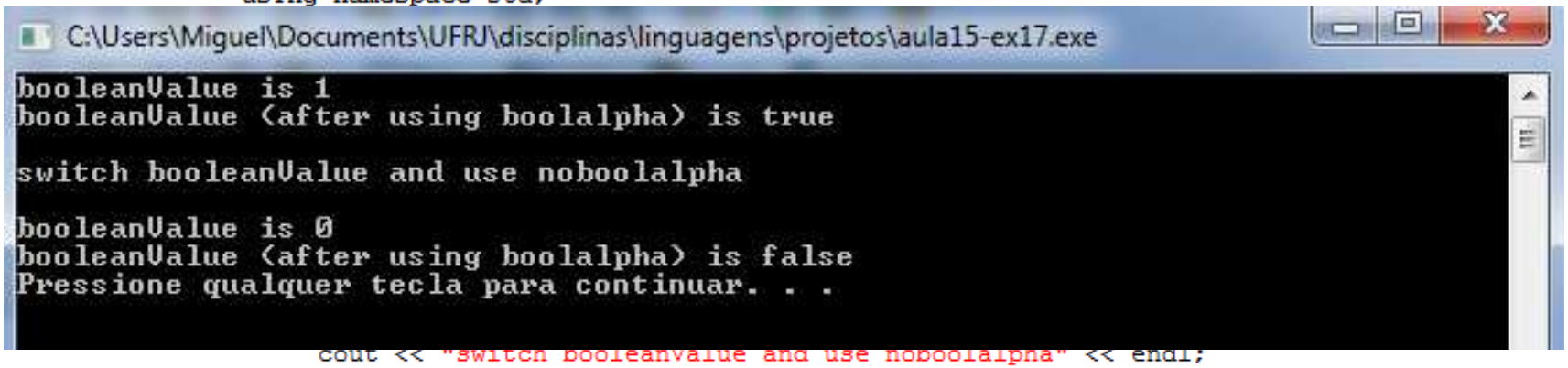
- Manipulador de fluxo **boolalpha**
  - Exibe strings **"true"** e **"false"**
  - Reiniciado com **noboolalpha**

Décimo Sétimo Exemplo Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 17
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
bool booleanValue = true;
     // exibe booleanValue true padrão
     cout << "booleanValue is " << booleanValue << endl;

     // exibe booleanValue depois de usar boolalpha
     cout << "booleanValue (after using boolalpha) is "
          << boolalpha << booleanValue << endl << endl;

     cout << "switch booleanValue and use noboolalpha" << endl;
     booleanValue = false; // muda booleanValue
     cout << noboolalpha << endl; // usa noboolalpha

     // exibe booleanValue false padrão depois de usar noboolalpha
     cout << "booleanValue is " << booleanValue << endl;

     // exibe booleanValue depois de usar after using boolalpha de novo
     cout << "booleanValue (after using boolalpha) is "
          << boolalpha << booleanValue << endl;

     return 0;
}
```

Décimo Sétimo Exemplo Usando E/S de Fluxo de Dados

```
/*
 * Aula 15 - Exemplo 17
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iomanip>

using namespace std;
```



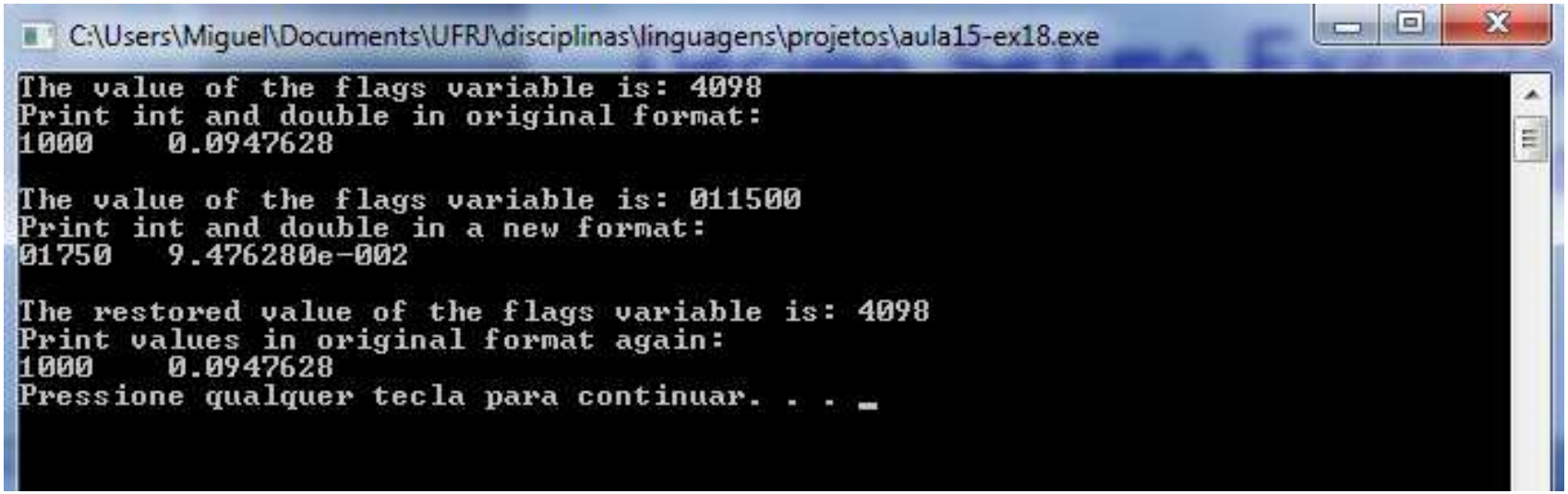

```
    cout << "switch booleanValue and use noboolalpha" << endl;
    booleanValue = false; // muda booleanValue
    cout << noboolalpha << endl; // usa noboolalpha

    // exibe booleanValue false padrão depois de usar noboolalpha
    cout << "booleanValue is " << booleanValue << endl;

    // exibe booleanValue depois de usar after using boolalpha de novo
    cout << "booleanValue (after using boolalpha) is "
         << boolalpha << booleanValue << endl;

    return 0;
}
```

Atribuição e Reatribuição de Formato via Função `flags`

- Pode salvar/restaurar estados de formatos
  - Depois de aplicar muitas mudanças, pode desejar restaurar formato original
- Função membro **`flags`**
  - **`cout.flags()`**
  - Sem argumento
    - Retorna estado corrente como objeto **`fmtflags`**
      - Namespace **`ios_base`**
    - Representa estado do formato
  - Com argumento **`fmtflags`**
    - Ajusta estado
    - Retorna o estado anterior

```cpp
/*
 * Aula 15 - Exemplo 18
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>

using namespace std;

int main() {
    int integerValue = 1000;
    double doubleValue = 0.0947628;

    // exibe valor de flags, valores int e double (formato original)
    cout << "The value of the flags variable is: " << cout.flags()
         << "\nPrint int and double in original format:\n"
         << integerValue << '\t' << doubleValue << endl << endl;

     // usa cout função flags para salvar formato original
     ios_base::fmtflags originalFormat = cout.flags();
     cout << showbase << oct << scientific; // muda formato

     // exibe valor de flags, valores int e double (novo formato)
     cout << "The value of the flags variable is: " << cout.flags()
          << "\nPrint int and double in a new format:\n"
          << integerValue << '\t' << doubleValue << endl << endl;

     cout.flags( originalFormat ); // reestabelece formato

     // exibe valores de flags, valores int e double values (formato original)
     cout << "The restored value of the flags variable is: "
          << cout.flags()
          << "\nPrint values in original format again:\n"
          << integerValue << '\t' << doubleValue << endl;

     return 0;
}
```

```
/*
 * Aula 15 - Exemplo 18
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>

using namespace std;

int main() {
    int integerValue = 1000;
    double doubleValue = 0.0947628;
```




```
    cout.flags( originalFormat ); // reestabelece formato

    // exibe valores de flags, valores int e double values (formato original)
    cout << "The restored value of the flags variable is: "
         << cout.flags()
         << "\nPrint values in original format again:\n"
         << integerValue << '\t' << doubleValue << endl;

    return 0;
}
```

Estado de Erro dos Fluxos

- Testa estado do fluxo usando bits
  - **`eofbit`** ligado quando EOF encontrado
    - Função **`eof`** retorna **`true`** se **`eofbit`** estiver ligado
    - **`cin.eof()`**
  - **`failbit`** ligado quando erro ocorre no fluxo
    - Dados não perdidos, erro é recuperável
      - Ex.: Programa está recebendo inteiros e um não inteiro é recebido
    - Função **`fail`** retorna **`true`** se ligado

Estado de Erro dos Fluxos

- Testa estado do fluxo usando bits
  - **badbit** ligado quando dados são perdidos
    - Usualmente não recuperável
    - Função **bad**
  - **goodbit** ligado quando **badbit**, **failbit** e **eofbit** estão desligados
    - Função **good**

Estado de Erro dos Fluxos

- Funções membro
  - `rdstate()`
    - Retorna estado de erro do fluxo
    - Pode testar por **`goodbit`**, **`badbit`** etc.
    - Melhor testar usando **`good()`**, **`bad()`** etc
      - **Assim programados não precisa conhecer todos os bits**

Estado de Erro dos Fluxos

- Funções membro
  - `clear()`
    - Argumento padrão **`goodbit`**
    - Atribui estado "good" ao fluxo, então I/O pode continuar
    - Pode passar outros valores
      - **`cin.clear( ios::failbit )`**
      - Atribui **`failbit`**
      - Nome "clear" parece estranho, mas correto

Décimo Nono Exemplo Usando E/S de Fluxo de Dados

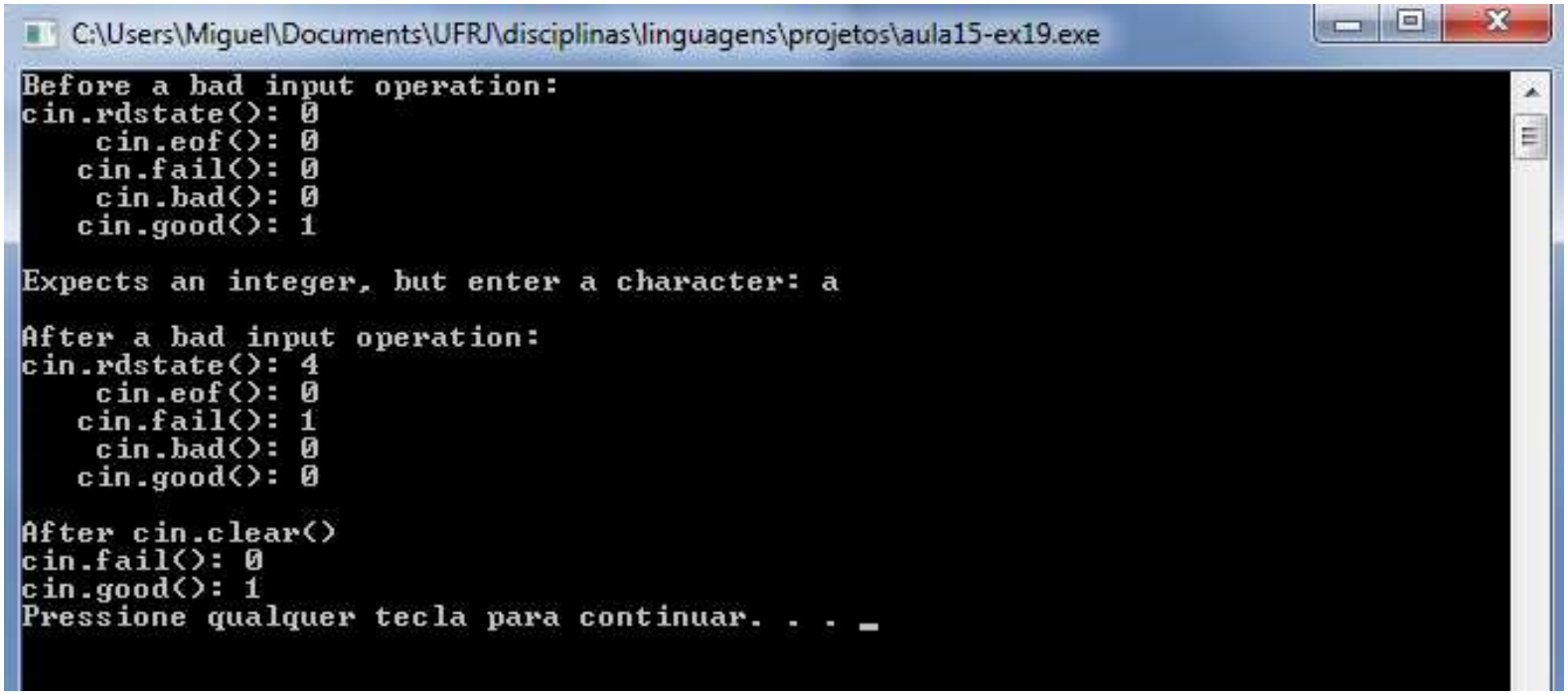
```
/*
 * Aula 15 - Exemplo 19
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>

using namespace std;

int main() {
    int integerValue;

    // exibe resultado da função cin
    cout << "Before a bad input operation:"
         << "\ncin.rdstate(): " << cin.rdstate()
         << "\n    cin.eof(): " << cin.eof()
         << "\n   cin.fail(): " << cin.fail()
         << "\n    cin.bad(): " << cin.bad()
         << "\n   cin.good(): " << cin.good()
         << "\n\nExpects an integer, but enter a character: ";

    cin >> integerValue; // entra valor do caractere
    cout << endl;

    // exibe resultados das funções cin depois da entrada ruim
    cout << "After a bad input operation:"
         << "\ncin.rdstate(): " << cin.rdstate()
         << "\n    cin.eof(): " << cin.eof()
         << "\n   cin.fail(): " << cin.fail()
         << "\n    cin.bad(): " << cin.bad()
         << "\n   cin.good(): " << cin.good() << endl << endl;
```

Décimo Nono Exemplo Usando E/S de Fluxo de Dados

```
    cin.clear(); // reinicia fluxo

    // exibe resultados das funções cin depois de reinicia cin
    cout << "After cin.clear()"
         << "\ncin.fail(): " << cin.fail()
         << "\ncin.good(): " << cin.good() << endl;

    return 0;
}
```

Décimo Nono Exemplo Usando E/S de Fluxo de Dados

```
cin.clear(); // reinicia fluxo
```



Amarrando uma Saída de Fluxo a uma Entrada de Fluxo

- Problema com saída em buffer
  - Programa interativo (prompt de usuário, ele/ela responde)
  - Prompt precisa aparecer antes da entrada proceder
    - Saídas em buffer aparecem apenas quando buffer encher ou for liberado

Amarrando uma Saída de Fluxo a uma Entrada de Fluxo

- Função membro **`tie`**
  - Sincroniza fluxos
  - Saídas aparecem antes de entradas subsequentes
  - Automaticamente realizado por **`cin`** e **`cout`**, mas poderia escrever
    - **`cin.tie( &cout )`**
  - Precisa amarrar explicitamente outros pares de I/O
  - Para desamarrar
    - **`inputStream.tie( 0 )`**

Leitura Recomendada

- Capítulos 15 do livro
  - Deitel, "*C++ How to Program*", 5th edition, Editora Prentice Hall, 2005