INF 213
Tratamento de exceções

Tratamento de exceções

- Uma exceção serve para indicar que um problema ocorreu durante a execução de um programa.
- Exceção: ocorre raramente (ou seja, é uma "exceção" à regra de execução típica do programa).
- O tratamento de exceções serve para se desenvolver programas capazes de resolver tais problemas.
- Exemplo, a função de inserção num *Conjunto* representado usando um array (como visto na aula prática).

Tratamento de exceções

- Neste caso, há duas situações "excepcionais" que impedem a inserção de um elemento no *Conjunto:*
  - Quando o elemento já existe no *Conjunto;*
  - Quando o número de elementos do conjunto alcança capacidade do array;
- No primeiro caso, faz sentido informar ao usuário que o elemento não pode ser inserido no array;
- Mas o segundo caso é um "problema devido à implementação" que deveria ser transparente ao usuário;

Tratamento de exceções

- Outro exemplo, dada uma classe *Aluno* contendo *matrícula*, *nome* e *cr*, suponha que você deseja escrever uma função que:
  - receba um array de alunos e uma matrícula
  - retorna uma referência para o objeto *Aluno* que possui a matrícula dada (se existir este *Aluno* é claro)

- A questão é: o que a função deve retornar caso o *Aluno* com a matrícula desejada não exista?

Tratamento de exceções

- Códigos que utilizam técnicas primitivas para tratamento de erros normalmente apresentam o seguinte aspecto:

  Realize uma tarefa

  Se a tarefa não foi executada de forma adequada

    Realize o processamento de erro

  Realize a próxima tarefa

  Se a tarefa anterior não foi executada de forma adequada

    Realize o processamento de erro

Tratamento de exceções

- Uma forma mais “elegante” de tratar tais erros consiste no uso do mecanismo de “tratamento de exceções”, que está presente em muitas linguagens de programação modernas.
- Para isso, definem-se classes que irão representar as exceções.
- As funções onde os erros podem ocorrer “lançarão” exceções (ou seja, indicarão erros) quando tais problemas ocorrerem.
- Cada trecho de código que utiliza tais funções deverá, por sua vez, apresentar trechos para monitorar e tratar tais exceções.

```
// Esta classe poderia ter qualquer nome
class Excecao {
    // Objeto a ser “lançado” para indicar a ocorrência de uma exceção
};
class Classe {
        public:
            void metodo();
        private:
};
void Classe::metodo() {
        throw Excecao();  // lança uma exceção (um objeto da classe Excecao)
}


int main() {
        Classe x;
        try {
            cout << "Ola!"<< endl;
            x.metodo();
            cout << "Ola 2" << endl;
        } catch(Excecao ex) {    // Captura a exceção (caso ela tenha sido lançada)
            cout << "Tratamento..." << endl;
        }
        cout << "fim!" << endl;
        return 0;
}
```

```
// Esta classe poderia ter qualquer nome
class Excecao {
    // Objeto a ser “lançado” para indicar a ocorrência de uma exceção
};
class Classe {
        public:
            void metodo();
        private:
};
void Classe::metodo() {
        throw Excecao();  // lança uma exceção (um objeto da classe Excecao)
}


int main() {
        Classe x;
        try {
            cout << "Ola!"<< endl;
            x.metodo();
            cout << "Ola 2" << endl;
        } catch(Excecao ex) {    // Captura a exceção (caso ela tenha sido lançada)
            cout << "Tratamento..." << endl;
```

```
$./a.out
./a.out
Ola!
Tratamento...
fim!
```

Tratamento de exceções

- As exceções são classes da linguagem C++.
- As funções que lançam exceções possuem uma sintaxe especial para declarar a possibilidade desse lançamento.
  - **Em C++, essa declaração é opcional** !
- Os trechos de código que tratam as exceções são circundados por um bloco “try { … } catch()”.
- Quando ocorre um lançamento de exceção (utilizando o comando *throw*), essa exceção “percorre” a pilha de chamadas de métodos até chegar em um bloco “protegido contra a exceção”.

Tratamento de exceções

- Ao chegar em um bloco protegido, a execução é direcionada para os *catchs*. A seguir, cada *catch* é analisado (de forma sequencial) para verificar se há um casamento entre o tipo da exceção lançada e o tipo do *catch*. Se houver um casamento, a exceção é "capturada".
- Ao ser capturada por um catch, **apenas** esse catch será executado e, após o término dele, a execução continuará depois do bloco protegido.
- Note que a exceção só é capturada se houver um catch específico para ela!

```
// Esta classe poderia ter qualquer nome
]class Excecao {
    // Objeto a ser “lançado” para indicar a ocorrência de uma exceção
};
class Classe {
        public:
            void metodo();
        private:
};
void Classe::metodo() {
        throw Excecao();  // lança uma exceção (um objeto da classe Excecao)
}


int main() {
        Classe x;
        try {
            cout << "Ola!"<< endl;
            x.metodo();
            cout << "Ola 2" << endl;
        } catch(Excecao ex) {    // Captura a exceção (caso ela tenha sido lançada)
            cout << "Tratamento..." << endl;
        }
        cout << "fim!" << endl;
        return 0;
}
```

```
// Esta classe poderia ter qualquer nome
]class Excecao {
    // Objeto a ser “lançado” para indicar a ocorrência de uma exceção
};
class Classe {
        public:
            void metodo();
        private:
};
void Classe::metodo() {
        throw Excecao();  // lança uma exceção (um objeto da classe Excecao)
}


int main() {
        Classe x;
        try {
            cout << "Ola!"<< endl;
            x.metodo();
            cout << "Ola 2" << endl;
        } catch(Excecao ex) {    // Captura a exceção (caso ela tenha sido lançada)
            cout << "Tratamento..." << endl;
```

```
$./a.out
./a.out
Ola!
Tratamento...
fim!
```

```
class Excecao {  };
class Excecao2 {   };
class Excecao3{  };
class Classe {
        public:
            void metodo();
};
void Classe::metodo() {
        throw Excecao();
}
int main() {
        Classe x;
        try {
                cout << "Ola!"<< endl;
                x.metodo();
                cout << "Ola 2" << endl;
        } catch(Excecao2 ex) {
                cout << "Tratamento 2..." << endl;
        } catch(Excecao ex) {
                cout << "Tratamento ..." << endl;
        } catch(Excecao3 ex) {
                cout << "Tratamento 3..." << endl;
        }
        cout << "fim!" << endl;

        return 0;
}
```

```
class Excecao {  };
class Excecao2 {   };
class Excecao3{  };
class Classe {
        public:
            void metodo();
};
void Classe::metodo() {
        throw Excecao();
}
int main() {
        Classe x;
        try {
                cout << "Ola!"<< endl;
                x.metodo();
                cout << "Ola 2" << endl;
        } catch(Excecao2 ex) {
                cout << "Tratamento 2..." << endl;
        } catch(Excecao ex) {
                cout << "Tratamento ..." << endl;
        } catch(Excecao3 ex) {
                cout << "Tratamento 3..." << endl;
        }
        cout << "fim!" << endl;
```

```
$./a.out
./a.out
Ola!
Tratamento...
fim!
```

Tratamento de exceções

- Uma estrutura *catch(...)* pode ser utilizada para capturar qualquer exceção:

```
void Classe::metodo() {
        throw Excecao();
}
int main() {
        Classe x;
        try {
                cout << "Ola!"<< endl;
                x.metodo();
                cout << "Ola 2" << endl;
        } catch( ... ) {
                cout << "Tratamento de qualquer excecao.." << endl;
        }
        cout << "fim!" << endl;
```

Tratamento de exceções

- Uma estrutura *catch(...)* pode ser utilizada para capturar qualquer exceção:

```
void Classe::metodo() {
        throw Excecao();
}
int main() {
        Classe x;
        try {
                cout << "Ola!"<< endl;
                x.metodo();
                cout << "Ola 2" << endl;
        } catch( ... ) {
                cout << "Tratamento de qualquer excecao.." << endl;
        }
```

```
Ola!
Tratamento de qualquer excecao..
fim!
```

Tratamento de exceções

- Uma estrutura *catch(...)* pode ser utilizada para capturar qualquer exceção:

```
void Classe::metodo() {
        throw Excecao();
}
int main() {
        Classe x;
        try {
                cout << "Ola!"<< endl;
                x.metodo();
                cout << "Ola 2" << endl;
        } catch (Excecao e) {
                cout << “aqui “ << endl;
        }
        catch( ... ) {
                cout << "Tratamento de qualquer excecao.." << endl;
        }
        cout << "fim!" << endl;
```

Tratamento de exceções

- Uma estrutura *catch(...)* pode ser utilizada para capturar qualquer exceção:

```
void Classe::metodo() {
        throw Excecao();
}
int main() {
        Classe x;
        try {
                cout << "Ola!"<< endl;
                x.metodo();
                cout << "Ola 2" << endl;
        } catch (Excecao e) {
                cout << “aqui “ << endl;
        }
        catch( ... ) {
```

```
Ola!
aqui
fim!
```

Tratamento de exceções

- Uma estrutura *catch(...)* pode ser utilizada para capturar qualquer exceção:

```
void Classe::metodo() {
        throw Excecao();
}
int main() {
        Classe x;
        try {
                cout << "Ola!"<< endl;
                x.metodo();
                cout << "Ola 2" << endl;
        }
        catch( ... ) {
                cout << "Tratamento de qualquer excecao.." << endl;
        }
        catch (Excecao e) {
                cout << “aqui “ << endl;
        }

        out << "fim!" << endl;
```

Tratamento de exceções

- Uma estrutura *catch(...)* pode ser utilizada para capturar qualquer exceção:

```
void Classe::metodo() {
        throw Excecao();
}
int main() {
        Classe x;
        try {
                cout << "Ola!"<< endl;
                x.metodo();
                cout << "Ola 2" << endl;
        }
        catch( ... ) {
                cout << "Tratamento de qualquer excecao.." << endl;
        }
        catch (Excecao e) {
                cout << "aqui " << endl;
```

```
g++ Excecao3.cpp
Excecao3.cpp: In function 'int main()':
Excecao3.cpp:26:2: error: '...' handler must be the last handler for its try block [-
fpermissive]
```

Tratamento de exceções

- Se uma exceção não for capturada em um método, esse método será abortado e a exceção irá continuar percorrendo a "pilha" de chamada até chegar em um bloco protegido.

- Note também que se a exceção lançada não for capturada e "atingir" o método *main*() o programa será abortado.

```
.....
void funcao() {
        Classe x;

        try {
                cout << "Ola!"<< endl;
                x.metodo();
                cout << "Ola 2" << endl;
        }  catch(Excecao ex) {
                cout << "Tratamento ..." << endl;
        }
        cout << "Aqui " << endl;

        return ;
}

int main() {
        Classe x;

        funcao();

        cout << "fim!" << endl;

        return 0;
}
```

```
.....
void funcao() {
        Classe x;

        try {
                cout << "Ola!"<< endl;
                x.metodo();
                cout << "Ola 2" << endl;
        }  catch(Excecao ex) {
                cout << "Tratamento ..." << endl;
        }
        cout << "Aqui " << endl;

        return ;
}

int main() {
        Classe x;

        funcao();

        cout << "fim!" << endl;

        return 0;
}
```

```
$./a.out
./a.out
Ola!
Tratamento...
Aqui
fim!
```

```
…..
void funcao() {
        Classe x;

        cout << "Ola!"<< endl;
        x.metodo();
        cout << "Ola 2" << endl;

        cout << "Aqui " << endl;

        return ;
}

int main() {
        Classe x;
        try {
                funcao();
        }  catch(Excecao ex) {
                cout << "Tratamento ..." << endl;
        }

        cout << "fim!" << endl;

        return 0;
}
```

```
…..
void funcao() {
        Classe x;

        cout << "Ola!"<< endl;
        x.metodo();
        cout << "Ola 2" << endl;

        cout << "Aqui " << endl;

        return ;
}

int main() {
        Classe x;
        try {
                funcao();
        }  catch(Excecao ex) {
                cout << "Tratamento ..." << endl;
        }

        cout << "fim!" << endl;

        return 0;
}
```

```
$./a.out
Ola!
Tratamento ...
fim!
```

```
.....
void funcao() {
        Classe x;

        cout << "Ola!"<< endl;
        x.metodo();
        cout << "Ola 2" << endl;

        cout << "Aqui " << endl;

        return ;
}

int main() {
        Classe x;
        try {
                funcao();
        }  catch(Excecao2 ex) {
                cout << "Tratamento ..." << endl;
        }

        cout << "fim!" << endl;

        return 0;
}
```

```
…..
void funcao() {
        Classe x;

        cout << "Ola!"<< endl;
        x.metodo();
        cout << "Ola 2" << endl;

        cout << "Aqui " << endl;

        return ;
}

int main() {
        Classe x;
        try {
                funcao();
        }  catch(Excecao2 ex) {
                cout << "Tratamento ..." << endl;
        }

        cout << "fim!" << endl;

        return 0;
}
```

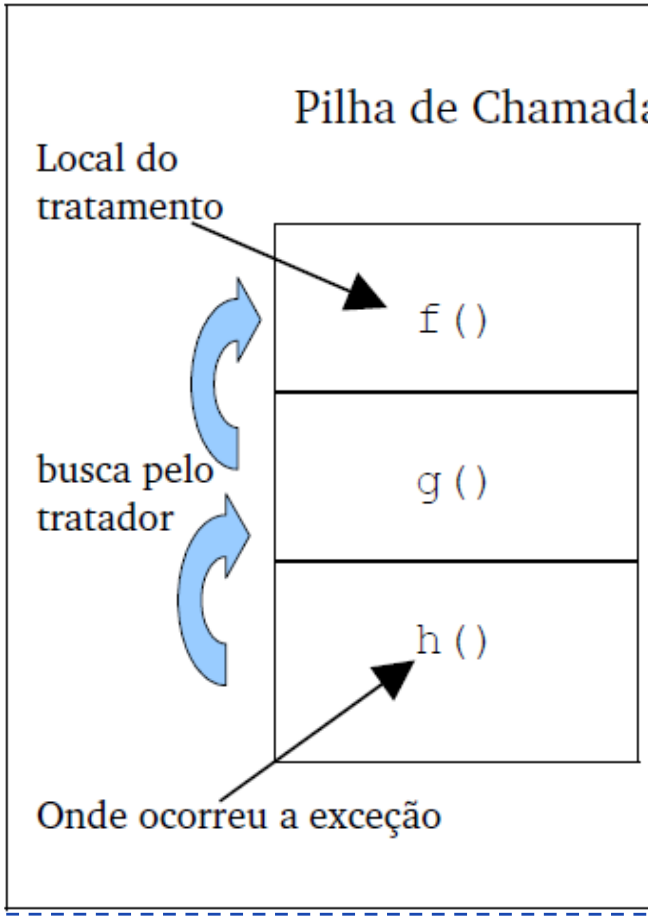
```
$./a.out
Ola!
terminate called after throwing an instance of 'Excecao'
Aborted
```

Tratamento de exceções

- Uma exceção pode ser qualquer objeto (pode ser inclusive tipos primitivos).
- Assim, ao lançar uma exceção estamos "enviando" um objeto para indicar a ocorrência de um erro.
- Esse objeto pode ter (e normalmente tem) valores.
- Tais valores normalmente são utilizados para facilitar a identificação/tratamento do erro.

Tratamento de exceções

- A figura abaixo mostra a sequência da busca por uma cláusula *catch* na pilha de chamadas.




```
void f(){
   try { g();}
   catch(exc1)
   {...}
}

void g()
{
   h(0);
   ...
}

void h(int p)
{
   if (p==0) throw exc1();
   ...
}
```

```
class Excecao {
        public:
                Excecao(char *m) {
                        msg = new char[strlen(m)+1];
                        strcpy(msg, m);                 }
                char *getMessage() {
                        return msg;       }
                //Faltam operador de atrib, constr. Copia, destrutor...
        private:
                char *msg;
};
class Classe {
        public:
                void metodo();
};
void Classe::metodo() {
        throw Excecao("houve um erro!");
}
int main() {
        Classe x;
        try {
                x.metodo();
        // Como nao implementamos o constr. de cópia/op. de atribuicao,
        // vamos passar a excecao por referencia..
        }  catch(Excecao &ex) {
                cout << "Ocorreu o seguinte erro: " << ex.getMessage() << endl;
        }
        cout << "fim!" << endl;
        return 0;
}
```

```
class Excecao {
        public:
                Excecao(char *m) {
                        msg = new char[strlen(m)+1];
                        strcpy(msg, m);                 }
                char *getMessage() {
                        return msg;       }
                //Faltam operador de atrib, constr. Copia, destrutor...
        private:
                char *msg;
};
class Classe {
        public:
                void metodo();
};
void Classe::metodo() {
        throw Excecao("houve um erro!");
}
int main() {
        Classe x;
        try {
                x.metodo();
        // Como nao implementamos o constr. de cópia/op. de atribuicao,
        // vamos passar a excecao por referencia..
        }  catch(Excecao &ex) {
                cout << "Ocorreu o seguinte erro: " << ex.getMessage() << endl;
        }
```

```
$ ./a.out
Ocorreu o seguinte erro: houve um erro!
Fim!
```

Tratamento de exceções

- Uma exceção capturada pode ser relançada para atingir funções "pai" na pilha de chamada.
- Para isso, basta utilizar o comando "*throw;*"
- Veja os exemplos a seguir.;

```
void funcao() {
	Classe x;
	cout << 3 << endl;
	try {
		cout << 4 << endl;
		x.metodo(); //Metodo que lança exceção
		cout << 5 << endl;
	} catch(Excecao &e) {
		cout << "Capturei a excecao!" << endl;

	}
}
int main() {
	cout << 1 << endl;
	try {
		cout << 2 << endl;
		funcao();
		cout << 6 << endl;
	}  catch(Excecao &ex) {
		cout << "Capturei a excecao 2!" << endl;
	}
	cout << "fim!" << endl;
```

```
void funcao() {
        Classe x;
        cout << 3 << endl;
        try {
                cout << 4 << endl;
                x.metodo(); //Metodo que lança exceção
                cout << 5 << endl;
        } catch(Excecao &e) {
                cout << "Capturei a excecao!" << endl;

        }
}
int main() {
        cout << 1 << endl;
        try {
                cout << 2 << endl;
                funcao();
                cout << 6 << endl;
        }  catch(Excecao &ex) {
                cout << "Capturei a excecao 2!" << endl;
        }
        cout << "fim!" << endl;
```

```
1
2
3
4
Capturei a excecao!
6
fim!
```

Tratamento de exceções

- Neste primeiro exemplo não houve a propagação da exceção;

- Agora veja o “mesmo programa” incluindo a propagação de uma exceção lançada;

```
void funcao() {
	Classe x;
	cout << 3 << endl;
	try {
		cout << 4 << endl;
		x.metodo(); //Metodo que lança exceção
		cout << 5 << endl;
	} catch(Excecao &e) {
		cout << "Capturei a excecao!" << endl;
		throw; //relança a excecao...
	}
}
int main() {
	cout << 1 << endl;
	try {
		cout << 2 << endl;
		funcao();
		cout << 6 << endl;
	}  catch(Excecao &ex) {
		cout << "Capturei a excecao 2!" << endl;
	}
	cout << "fim!" << endl;
```

```
void funcao() {
        Classe x;
        cout << 3 << endl;
        try {
                cout << 4 << endl;
                x.metodo(); //Metodo que lança exceção
                cout << 5 << endl;
        } catch(Excecao &e) {
                cout << "Capturei a excecao!" << endl;
                throw; //relança a excecao...
        }
}
int main() {
        cout << 1 << endl;
        try {
                cout << 2 << endl;
                funcao();
                cout << 6 << endl;
        }  catch(Excecao &ex) {
                cout << "Capturei a excecao 2!" << endl;
        }
        cout << "fim!" << endl;
```

```
1
2
3
4
Capturei a excecao!
Capturei a excecao 2!
fim!
```

Tratamento de exceções

- A **estrutura** de tratamento de exceções gera pouca ou nenhuma penalidade de desempenho nos programas.
- Assim, programas que utilizam esta técnica podem ser mais eficientes do que programas que utilizam códigos de erro (os que utilizam código são penalizados pelo uso de vários “ifs”).
- Porém, o tratamento de exceções não deve ser utilizado como uma forma de controle do fluxo dos programas pois:
  - As exceções adicionais podem ser confundidas com verdadeiras exceções.
  - O **lançamento** de exceções não é eficiente.

Tratamento de exceções

- O operador *new* do C++ pode lançar exceções.
- Mais especificamente, em caso de falhas são lançadas exceções do tipo “bad_alloc”.
- Veja os exemplos a seguir:

```
int main() {
	int *v;

	for(int i=0;i<10;i++) {
		long long numElementos = i*1024*1024*(1024ll);

		cout << "Tentando alocar: " << (1.0*numElementos*sizeof(int))/(1024*1024*1024)
<< " GBs de memória" <<endl;
		v = new int[numElementos];
		cout << "Memória alocada " << endl;
		delete []v;
		cout << "Memória desalocada" << endl;
	}

	return 0;
}
```

```
int main() {
        int *v;

        for(int i=0;i<10;i++) {
                long long numElementos = i*1024*1024*(1024ll);

                cout << "Tentando alocar: " << (1.0*numElementos*sizeof(int))/(1024*1024*1024)
<< " GBs de memória" <<endl;
                v = new int[numElementos];
                cout << "Memória alocada " << endl;
                delete []v;
                cout << "Memória desalocada" << endl;
        }

        return 0;
```

```
$ ./a.out
Tentando alocar: 0 GBs de memória
Memória alocada
Memória desalocada
Tentando alocar: 4 GBs de memória
Memória alocada
Memória desalocada
Tentando alocar: 8 GBs de memória
Memória alocada
Memória desalocada
Tentando alocar: 12 GBs de memória
terminate called after throwing an instance of 'std::bad_alloc'
  what():  std::bad_alloc
Aborted
```

```
int main() {
	int *v;

	for(int i=0;i<10;i++) {
		long long numElementos = i*1024*1024*(1024ll);

		cout << "Tentando alocar: " << (1.0*numElementos*sizeof(int))/(1024*1024*1024)
<< " GBs de memória" <<endl;
		try {
			v = new int[numElementos];
			cout << "Memória alocada " << endl;
			delete []v; //Porque o delete tem que estar dentro do try...catch ?
			cout << "Memória desalocada" << endl;
		} catch (bad_alloc excecao) {
			cout << "Problema ao alocar memória... " <<endl;
		}
…..
```

```
int main() {
	int *v;

	for(int i=0;i<10;i++) {
		long long numElementos = i*1024*1024*(1024ll);

		cout << "Tentando alocar: " << (1.0*numElementos*sizeof(int))/(1024*1024*1024)
<< " GBs de memória" <<endl;
		try {
			v = new int[numElementos];
			cout << "Memória alocada " << endl;
			delete []v; //Porque o delete tem que estar dentro do try...catch ?
			cout << "Memória desalocada" << endl;
		} catch (bad_alloc excecao) {
			cout << "Problema ao alocar memória... " <<endl;
		}
```

```
Tentando alocar: 0 GBs de memória
Memória alocada
Memória desalocada
Tentando alocar: 4 GBs de memória
Memória alocada
Memória desalocada
Tentando alocar: 8 GBs de memória
Memória alocada
Memória desalocada
Tentando alocar: 12 GBs de memória
Problema ao alocar memória...
Tentando alocar: 16 GBs de memória
Problema ao alocar memória...
....
```

```
int main() {
        int *v;
        for(int i=0;i<10;i++) {
                long long numElementos = i*1024*1024*(1024ll);

                cout << "Tentando alocar: " << (1.0*numElementos*sizeof(int))/(1024*1024*1024)
<< " GBs de memória" <<endl;
                try {
                        v = new int[numElementos];
                        cout << "Memória alocada " << endl;
                        delete []v; //Porque o delete tem que estar dentro do try...catch ?
                        cout << "Memória desalocada" << endl;
                } catch (bad_alloc excecao) {
                        cout << "Problema ao alocar memória... " <<endl;
                        cout << excecao.what() << endl; //O bad_alloc possui o método “what”
                }
…..
```

```
int main() {
        int *v;
        for(int i=0;i<10;i++) {
                long long numElementos = i*1024*1024*(1024ll);

                cout << "Tentando alocar: " << (1.0*numElementos*sizeof(int))/(1024*1024*1024)
<< " GBs de memória" <<endl;
                try {
                        v = new int[numElementos];
                        cout << "Memória alocada " << endl;
                        delete []v; //Porque o delete tem que estar dentro do try...catch ?
                        cout << "Memória desalocada" << endl;
                } catch (bad_alloc excecao) {
                        cout << "Problema ao alocar memória... " <<endl;
                        cout << excecao.what() << endl; //O bad_alloc possui o método “what”
                }
```

```
Tentando alocar: 0 GBs de memória
Memória alocada
Memória desalocada
Tentando alocar: 4 GBs de memória
Memória alocada
Memória desalocada
Tentando alocar: 8 GBs de memória
Memória alocada
Memória desalocada
Tentando alocar: 12 GBs de memória
Problema ao alocar memória...
std::bad_alloc
Tentando alocar: 16 GBs de memória
....
```