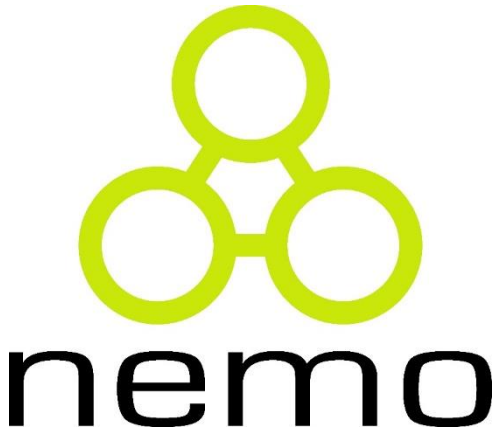


Programação III

Jordana S. Salamon

jssalamon@inf.ufes.br

jordanasalamon@gmail.com

DEPARTAMENTO DE INFORMÁTICA

CENTRO TECNOLÓGICO

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

Exceções

Definição

- Uma exceção é um problema que surge durante a execução de um programa.
- Em C++, uma exceção é uma resposta para um circunstância excepcional que surge enquanto um programa está executando, como por exemplo uma divisão por 0.

```
#include <iostream>

using namespace std;

int main(){
    cout << 20/0 << endl;
    return 0;
}
```

nemo

Manipulação de Exceções

- Exceções proveem uma forma de transferir o controle de uma parte do programa para outra sem que o programa seja finalizado com erro.
- Basicamente a manipulação de uma exceção em C++ é construída utilizando 3 palavras chaves:
  - **Throw:** um programa lança uma exceção quando um programa encontra um problema de execução, isto é feito utilizando a palavra **throw**.
  - **Catch:** um programa captura uma exceção com um manipulador de exceções no local onde o programa encontra um problema. A palavra **catch** indica a captura da exceção.
  - **Try:** um bloco de comando try representa um código que está preparado para ser ativado caso o programa identifique alguma exceção. Ele é seguido de um ou mais blocos catch.

Manipulação de Exceções

- Assumindo que um bloco do código pode surgir uma exceção, o método captura uma exceção utilizando uma combinação das palavras try e catch.
- Um bloco try/catch é colocado envolta do código que pode gerar um exceção. Desta forma o código dentro do try/catch é referido como um código protegido.

```
try
{
   // protected code
}catch( ExceptionName e1 )
{
   // catch block
}catch( ExceptionName e2 )
{
   // catch block
}catch( ExceptionName eN )
{
   // catch block
}
```

nemo

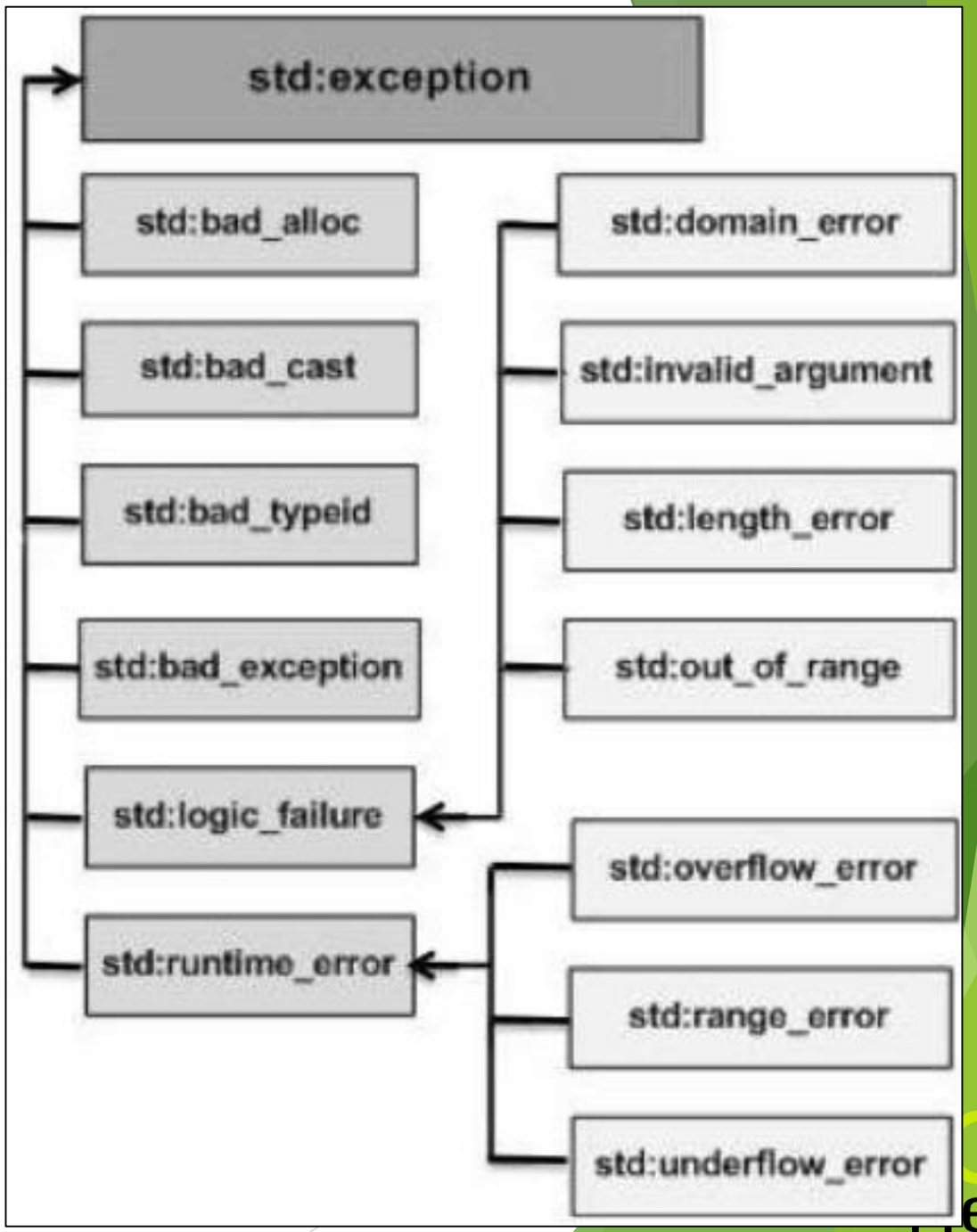
Exceções da Linguagem

```
int main(){
    vector<int> myvector(10);

    try
    {
        int* myarray= new int[10000000000000000];

        bitset<5> mybitset (string("01234"));

        myvector.at(20)=100;
    }
    catch (const std::bad_alloc& ba)
    {
        cerr << "Alocacao de Memoria Indevida (" << ba.what() << ")"<< endl;
    }
    catch (const std::invalid_argument& ia)
    {
        cerr << "Argumentos Invalidos (" << ia.what() << ")"<< endl;
    }
    catch (const std::out_of_range& oor)
    {
        cerr << "Acesso de posicao indevida (" << oor.what() << ")"<< endl;
    }
    return 0;
}
```

Exceções da Linguagem

Biblioteca de Exceções:
<stdexcept>

Leia mais em:
http://www.cplusplus.com/reference/exception/exception/?kw=exception



Tratando Exceções

```
int main(){
    int i;
    vector<int> v;
    for(i=0;i<10;i++){
        v.push_back(i+1);
    }
    int n;
    cout << "Digite um numero n: ";
    cin >> n;

    try
    {
        cout << "O valor na posicao " << n << " = " << v.at(n-1) << endl;
    }
    catch (const std::out_of_range& oor)
    {
        cerr << "A posicao " << n << " nao existe no vetor" << endl;
        cout << "Digite um novo numero n: ";
        cin >> n;
        cout << "O valor na posicao " << n << " = " << v.at(n-1) << endl;
    }
    return 0;
}
```

nemo

Criando Exceções Personalizadas

- A linguagem permite que sejam lançadas exceções personalizadas utilizando o comando **throw**, e essa exceções personalizada pode ser capturada em algum local do código.

```
#include <stdexcept>

using namespace std;

double divisao(int a, int b)
{
    if(b==0)
    {
        throw "Nao eh permitido divisao por 0";
    }
    return (a/b);
}
```

```
int main(){
    int x;
    int y;
    double z;

    cout << "Digite o numerador: ";
    cin >> x;
    cout << "Digite o denominador: ";
    cin >> y;

    try {
        z = divisao(x, y);
        cout << "Resposta: " << z << endl;
    }catch (const char* msg) {
        cerr << msg << endl;
    }
    return 0;
}
```

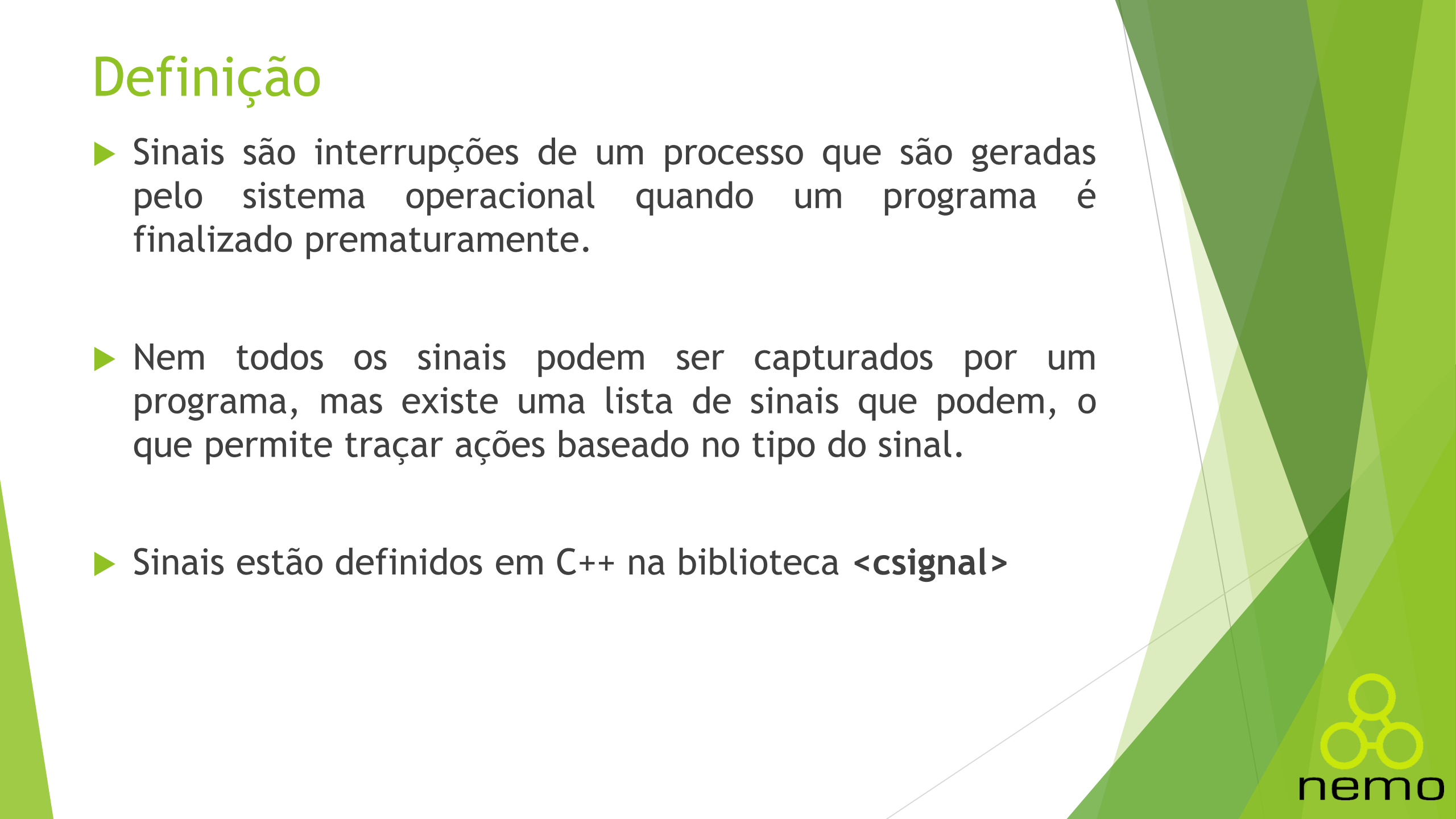
Sinais

Definição

- Sinais são interrupções de um processo que são geradas pelo sistema operacional quando um programa é finalizado prematuramente.

- Nem todos os sinais podem ser capturados por um programa, mas existe uma lista de sinais que podem, o que permite traçar ações baseado no tipo do sinal.

- Sinais estão definidos em C++ na biblioteca **<csignal>**



Tipos de Sinais

Signal	Description
SIGABRT	Abnormal termination of the program, such as a call to **abort**.
SIGFPE	An erroneous arithmetic operation, such as a divide by zero or an operation resulting in overflow.
SIGILL	Detection of an illegal instruction.
SIGINT	Receipt of an interactive attention signal.
SIGSEGV	An invalid access to storage.
SIGTERM	A termination request sent to the program.

nemo

Função Signal()

- A biblioteca prove um função para capturar os eventos inesperados, a função signal():

```
void (*signal (int sig, void (*func)(int)))(int);
```

- A função possui dois argumentos:
  - o primeiro é um inteiro, representando o tipo do sinal (e.g. SIGINT).
  - O segundo é um ponteiro para a sua função, criada para a manipulação dos sinais.
- A chamada da função deve ficar em qualquer lugar do código acima da possível linha que pode gerar um erro.

Função para manipulação dos sinais

```
void manipulacaoSinal(int sinal)
{
    switch (sinal){
        case SIGFPE:
        {
            cout << endl << "Divisao por 0!" << endl;
            break;
        }
        case SIGINT:
        {
            cout << endl << "Fim do Programa!" << endl;
            break;
        }
        case SIGABRT:
        {
            cout << endl << "Programa abortado!" << endl;
            break;
        }
        case SIGSEGV:
        {
            cout << endl << "Falha de Segmentacao!" << endl;
            break;
        }
    }
    exit(sinal);
}
```

nemo

Chamando a função std::signal

```
double divisao(int a, int b)
{
    signal(SIGFPE, manipulacaoSinal);
    return (a/b);
}
```

Código do Sinal

Nome da minha função

nemo

Função Raise()

- A biblioteca prove um função também para gerar sinais, a função raise():

```
int raise (signal sig);
```

- O argumento passado para a função é o sinal que você deseja gerar e o valor de retorno é o número do sinal.

That’s all Folks!