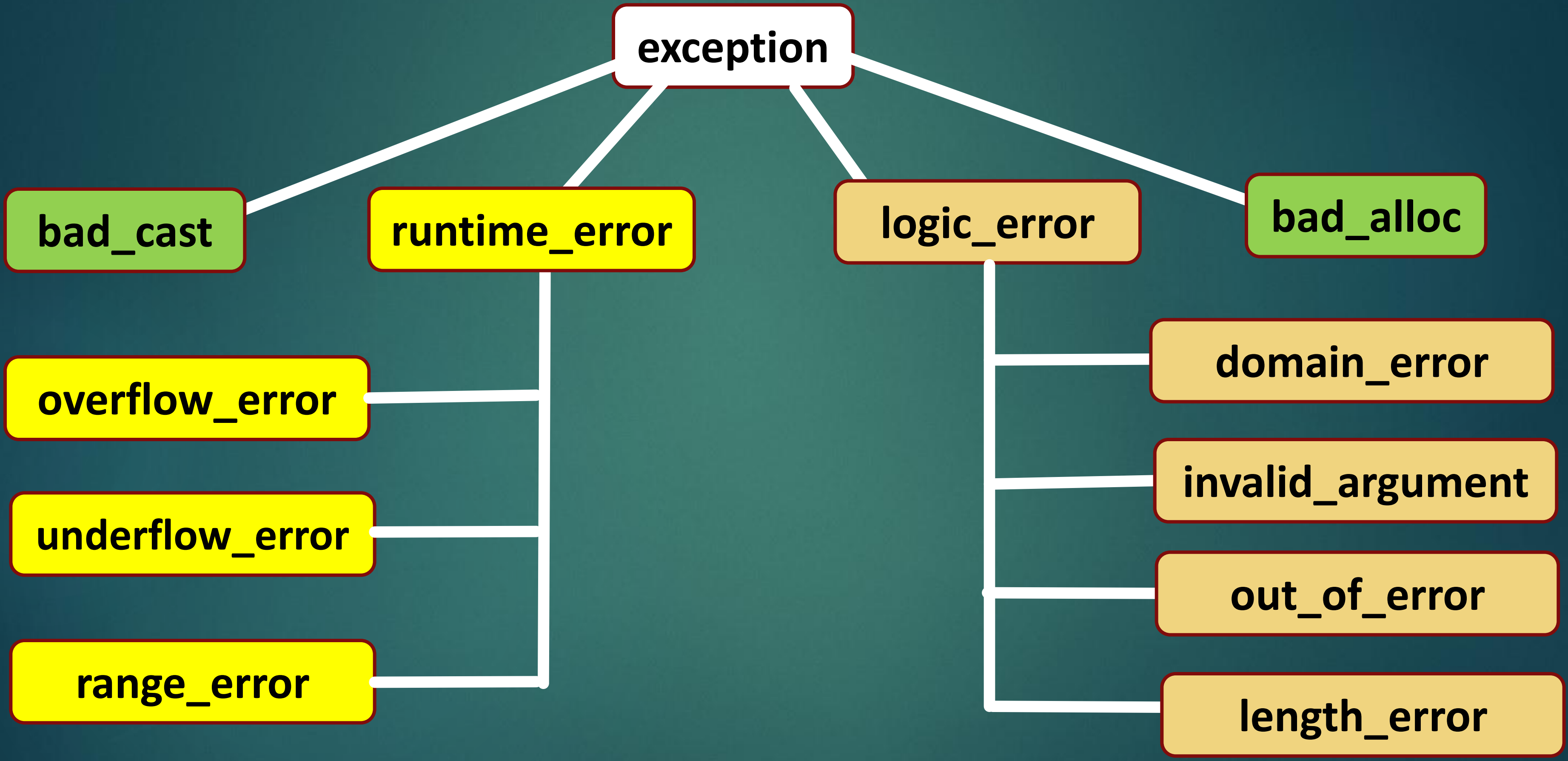
Exception Handling

Exceptions

- An **exception** is a abnormal condition that occurs at run time and disturbs normal continuation of the program
- When an exception occurs, the program must either terminate or jump to exception handling code
- The special code for exception handling the is called an **Exception Handler**
- **Example: Divide by Zero, Array Out Of Bound etc.**

Exception Hierarchy in C++



Header files for using exceptions

Exception classes	Header file required to be included
exception	#include<exception>
runtime_error, logic_error & their subclasses	#include<stdexcept>
bad_alloc	#include<new>
bad_cast	#include<type_info>

Exceptions Handling - keywords

- **throw** – throw is used to throw the exception instead of handling it using **try** and **catch**
- **try** – try block is used to invoke code that may throw or throws an exception
- **catch** – catch block is used to handle exceptions thrown in preceding try block.
    - In c++ we can throw any data type variable
    - **catch** block can not be written without **try block**

try, catch & throw example

```
#include<iostream>
#include<stdexcept>
using namespace std;
float Divide(int a, int b){
        float ans;
        if(b ==0){
throw runtime_error("Divide by Zero Error");
        }
        else{
                ans =a/b;
        }
}
```

```
int main(){
try{
float avg = Divide(10,0);
cout<<"Average="<<avg;
}
catch(runtime_error e){
cout<<e.what();
}
return 0;
}
```

What can be thrown and caught?

- In C++, throw block can throw & catch block can catch any variable/object of any data type
- throw clause example
  - throw “Emergency!”;
  - throw 12;
  - throw runtime_error(“runtime error occurred”);
- catch block example

```
catch(char *s){
//handling }
```

```
catch(int i){
//handling }
```

```
catch(runtime_error e){
//handling }
```

Exception handling mechanism

- Computer encounters a **throw** statement in a **try** block
- The computer evaluates the **throw** expression, and immediately exits the **try** block
- The computer selects an attached **catch** block that matches the type of the thrown value, places the value in the catch block's formal parameter, and executes the catch block

Unhandled Exceptions

- An unhandled exception propagates backwards into the calling function and appears to be thrown at the point of the call
- The computer will keep terminating function calls and tracing backwards along the call chain until it finds an enclosing **try** block with a matching handler, or until the exception propagates out of **main** (terminating the program).
- This process is called **Stack Unwinding**

Handling Multiple Exceptions

- Multiple catch blocks can be attached to the same block of code. The catch blocks should handle exceptions of different types

```
try{...}
 catch(int iEx){ }
 catch(char *strEx){ }
 catch(double dEx){ }
```

Generic catch block

- C++ allow user to write generic catch block to handle all types of exception.

Example:

```
catch(…)
{
//This catch block can handle any exception thrown
}
```

Custom Exception

- Programmer can create custom exception by inheriting any built-in exception class

```
#include<iostream>
#include<stdexcept>
using namespace std;
Class MyException : public runtime_error
{
//Customized functions and overridden functions.
}};
```

throw clause

- Programmer can specify exceptions that function in throwing using throw clause

```
void function1() throw (runtime_error)
{
// code that throw exception
}
```

Thank You

Never ignore me!!!!!

Eating exception, makes fat programmer !!!

……….Exception