Advance Programming in C/C++ - Homework 5 - Exercise 3

Alessandro Castelli ID: 12246581

4 December, 2023

1 Exercise 3

1.1 Exception in C++

`List and describe at least 5 standard C++ exceptions. Explain if other languages (such as Java) support exceptions and how they are integrated within the language.`

1. **runtime_error**: Is an error derived from the `Exception` class and is thrown as an exception when problems occur during runtime, such as:

   - Memory allocation error
   - Dynamic casting error
   - Overflow errors
   - Underflow errors
   - Division by zero

   The `runtime_error` exception class also has some very important derived exception classes like: `overflow_error, underflow_error, range_error`. If the exception is not handled the program ends with an error message.

   Example: C++ runtime_error code

   ```
   class Calculator {
       public:
           double divide(double a, double b) {
               if (b == 0.0) {
                   throw std::runtime_error("Error: division by 0");
               }
               return a / b;
           }
   };
   ```

2. **logic_failure**: This is a standard exception class in C++ that represents a logical error in the code. A `logic_failure` error occurs when the code violates a logical condition that should always be true, such as a contract or an invariant that must always hold. This error is not dependent on external factors but solely on the program's logic. If the exception is not handled, the program terminates with an error message. `logic_failure` has four standard derived classes, which are: `domain_error, invalid_argument, length_error, and out_of_range`.

   Example: C++ logic_failure code

   ```
   int main() {
       std::vector<int> myVector = {1, 2, 3, 4, 5};

       try {
           int value = myVector.at(10);
       } catch (const std::out_of_range& e) {
           std::cerr << "Error: " << e.what() << std::endl;
       }
       return 0;
   }
   ```

3. **bad_alloc**: This is a standard exception that derives from the `Exception` class and is thrown when a memory allocation failure occurs. This error may arise, for example, when attempting to allocate memory for an array or, in general, for a data structure that exceeds the available memory on the machine at that moment. If the exception is not handled, the program terminates with an error message.

4. **bad_cast**: This is a standard exception that represents a failure in the dynamic cast of a reference to a polymorphic type. This class derives from the `Exception` class. If the exception is not handled, the program terminates with an error message.

Example: C++ bad_cast code

```
class Animal {virtual void member () {}}; // polymorphic base class
class Dog : Animal {}; // derived class
using namespace std;
int main () {
  try {
    Animal a;
    Dog& rd = dynamic_cast<Dog&> (a);
  }
  catch (bad_cast& bc) {
    cerr << "bad_cast caught:" << bc.what () << '\n';
  }
  return 0;
}
```

5. **bad_typeid**: This is also a standard exception in C++ that occurs when the `typeid` operator is used on a dereferenced null pointer of a polymorphic type. This class derives from the `Exception` class. If the exception is not handled, the program terminates with an error message.

Example: C++ bad_cast code

```
int main()
{
    int* p = nullptr;
    try
    {
        cout << "The type of *p is:" << typeid(*p).name() << endl;
    }catch (const bad_typeid& e){
        cerr << "Error bad_typeid:" << e.what() << endl;
    }

    return 0;
}
```

1.2 Exceptions in other languages

Many other programming languages also support exceptions, such as Java, C#, Python and JavaScript. All these programming languages use `try` and `catch` blocks, or similar structures, to handle exceptions.

- In Java, to generate an exception, the `throw` keyword is used. To catch an exception, the `try` and `catch` blocks are employed. The `try` block contains the code that might generate an exception, while the `catch` block specifies how to handle the exception. The `finally` block is used to specify code that must be executed regardless of whether an exception is present or not.

  Example: Java Exception

```java
public class Example {
    public static void main(String[] args) {
        try {
            // Code
        } catch (Exception e) {
            // Exception Management
        } finally {
            // Code
        }
    }
}
```

- In Python, to raise an exception, the keyword `raise` is used. To catch an exception, the block `try` and `except` is used. The block `try` contains the code that might raise an exception, while the block `except` specifies how to handle the exception. The block `else` is executed only if no exception is raised in the block `try`. The block `finally` contains the code that must be executed always, regardless of the presence or absence of an exception.

- in C# the structure `Try, Catch, Finally` is used, in which, the block `try` contains the instructions that can generate an exception. If an exception occurs in this block, the control flow passes to the first compatible `Catch` block. The block `Catch` contains the instructions to handle the exception. Multiple catch blocks can be used to intercept different types of exceptions. The catch block must specify the type of exception to catch, which must be derived from the Exception class. Filters can also be used to catch only the exceptions that satisfy a certain condition. The block finally contains the instructions that are executed in any case, whether or not an exception occurs in the try block.

As you can see from this brief explanation, the exception handling in different languages is very similar, although with some small differences regarding the syntax and the features.