Exception Handling

Abstract: (summary of work)

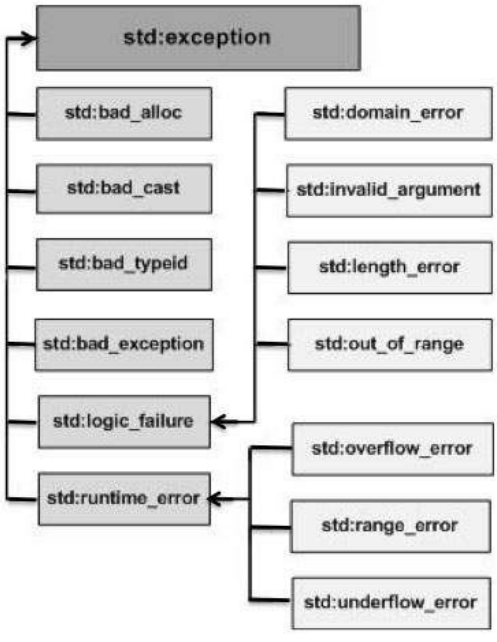
Using exceptions to deal with errors can make it easier for programmers to prevent errors within their code while keeping their code clean. Using if statements to detect/prevent errors in your code may be effective for small programs but with bigger blocks of codes comes more chances to create errors. All of those if statements will clutter and make your program confusing to read. That's where exceptions handling shines. In this project, we learned about the try and catch block method and throwing out exceptions for when an error may occur.

Intro:

When running a program you may come across an error that causes your program to fail. These errors can be anything from a missing semicolon to using the wrong data type. To detect and stop where these errors occur we use exceptions. Exceptions are run-time anomalies and abnormal conditions that occur during the execution of your program. There are two types of exceptions, Synchronous and Asynchronous, however, exception handling is only for synchronous exceptions. When you execute something in your code synchronously, you have to wait for it to finish before it can move onto the next problem. Asynchronously running something allows you to do other tasks while you are waiting for that task to complete itself.

RAII stands for Resource Acquisition Is Initialization, which is a technique primarily used in c++. RAII is a technique that binds the life cycle of a resource to a lifetime of an object. In an RAII class, the constructor acquires resources and establishes class invariants if that can't be done the constructor will throw an exception. The destructor will release the resource but it will not throw exceptions. Examples of RAII classes in the standard library are std::string, vector, thread, array, and mutex. RAII sometimes goes by SBRM(Scope-based Resource Management).

C++ has 3 keywords for programmers to make exceptions in their program. *Try, Catch*, and *Throw.* Try is a keyword you use on a block code that may cause an error. If an exception may occur it will be thrown from the try block. Catch is a block of code that corresponds with the try block of code. There can be more than one catch block depending on the number of exceptions that your try block identifies. Catch executes when a specific exception is caught by the system. Throw is a statement we can use anywhere within a block of code, no need for it to be within try. Throw is where we print our exceptions out onto the console. C++ has a list of standard exceptions in its library defined in <exception>.



Method:

To learn about our topic we first had to read about the exception class and the keywords we use for creating exceptions. In the exception class we can find almost every

error we may need. The exception class contains exceptions for runtime errors, bad allocation, stack over/underflow and many others. In order to learn how these errors occur we played around with std::exception and try, catch, throw statements.

Try, catch, throw, follows a basic format with a try encapsulating whatever line(s) of code you want to throw an exception in. The throw falls within the try block and will be called after whatever condition we set for there to be an error/exception. Catch is a block of code that immediately follows the try block. There can be more than one catch after the try depending on how many exceptions can be thrown from your try block. You will need a catch block per exception.

```
try{
    // code to be tested
throw exception;
}catch(type exception){
    // code to be executed
}
```

Aside from the exceptions given to us in the class, we can create our own exceptions by extending the exception class. To do this we must use the <exception> header. To make an exception you have to inherit and override the exception class. In this derived class that we create, we must override the what() function. what() returns a null terminated character that can be used to identify an exception. Following what() is where we throw our exception, after that your custom exception is ready to use. Here is the example we made with comments for what each step does.

```
#include <iostream>
#include <exception> // what()
using namespace std;

//Inheriting the exception class//
class MyException: public exception{
    public:
    //overiding what()//
        virtual const char* what() const throw(){
            //Our custom exception message//
            return "Oops, something went wrong!";
        }
};

class Test{
    public:
        void somethingWentWrong(){
            throw MyException();
        }
};

int main () {
  Test test;
  try{
      //try identifies any errors that may have occurred in test class//
      test.somethingWentWrong();
  }
  //checks to see if an error was caught by try block
  //comands our exception to run its what()//
  catch(MyException &e){
      cout << e.what() <<endl;
  }
  return 0;
}
```

RAII is a style of making your classes. When you make your class you want to make sure the constructors aquire resources and that the destructors release those resources. This is important for your program to make sure theres no memory leaks in your program. The object created from the class will acquire and remove the resources it needs during its lifetime so there's no need to remove those

```
class Speaker{
    private:
        std::string name;
    public:
        Speaker(){
            std::cout<< "Default constructing \"\""<<std::endl;
        }
        Speaker(std::string n) : name(n){
            std::cout<<" constructing \""<<name<<"\""<<std::endl;// constructor acquires resorce
        }
        ~Speaker(){
            std::cout<<"Destructing \""<<name<<"\""<<std::endl; //Destructor releases resource
        }

};
```

resources later on in your program. This removes having to remember to delete said resources later and saves you from writing extra code later on. Here is an example of a class made with RAII in mind.

Conclusion:

Prior to this project the main way we tested errors would be through the use of if statements. If statements may get the job done, but there are certain problems that arise when we solely use them. When a program only has a few lines of code if statements are easier to implement and there isn't really a need to use exception, but when a program has hundreds or thousands of lines, using an if statement is inefficient. We all learned the method of using try, catch, and throw blocks in order to optimize error outputs. We also learned the importance of using RAII in the creation of classes. One of the most important things this project taught us is collaboration and working as a group. Most of us were not very familiar with working as a group for an assignment of this type and found this experience beneficial. We now know the preparations we must take for our future projects and the expectations that will be made for us. Overall this project was a great learning experience for us all and we will be looking forward to the next one.

References

Taking advantage of RAII/SBRM

1. Johnston, P. (2019). *Migrating from C to C++: Take Advantage of RAII/SBRM — Embedded Artistry*. [online] Embedded Artistry. Available at: https://embeddedartistry.com/blog/2017/7/17/migrating-from-c-to-c-take-advantage-of-raiisbrm [Accessed 13 Oct. 2019].

RAII CPP Reference

2. En.cppreference.com. (2019). *RAII - cppreference.com*. [online] Available at: https://en.cppreference.com/w/cpp/language/raii [Accessed 13 Oct. 2019].

Exception and Error Handling

3. Isocpp.org. (2019). *Standard C++*. [online] Available at: https://isocpp.org/wiki/faq/exceptions [Accessed 13 Oct. 2019].

std::Exception C++ Reference

4. Cplusplus.com. (2019). *exception - C++ Reference*. [online] Available at: http://www.cplusplus.com/reference/exception/exception/ [Accessed 13 Oct. 2019].

Exceptions

5. Cplusplus.com. (2019). *Exceptions - C++ Tutorials*. [online] Available at: http://www.cplusplus.com/doc/tutorial/exceptions/ [Accessed 13 Oct. 2019].

Exception handling

6. Tutorialspoint.com. (2019). *C++ Exception Handling - Tutorialspoint*. [online] Available at: https://www.tutorialspoint.com/cplusplus/cpp_exceptions_handling.htm [Accessed 13 Oct. 2019].