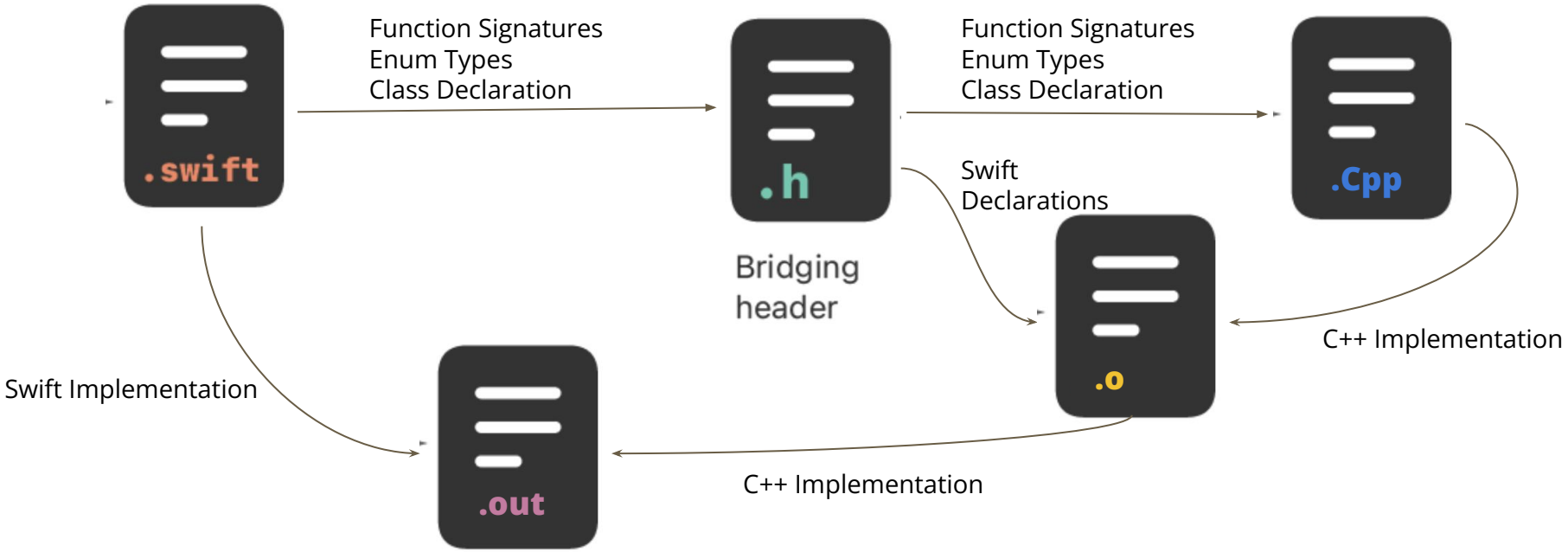
Bridging Swift Error Handling Model to C++

Roberto Gomes Rosmaninho Neto
Advisor: Fernando Magno Quintão Pereira

Context: Interoperability Between Swift and C++

- Functions written in Swift and used in C++



Contexto: Error Handling

- Swift Errors ≠ C++ Exceptions
- Erros lançados em Swift nunca eram tratados em C++

Contexto: Exemplo

```
public enum DivErrors : Error {
    case divByZero
}
```

Erro em Swift

```
public func divInt(num: Int, div: Int) throws -> Float {

}
```

Função que pode lançar um Erro

General Goal

- Modify the Swift compiler to recognize and represent a *Swift::Error* trew by a Swift function called by a C++ program.

- Allow handling of *Swift::Errors* thrown as exceptions using C++ try-catch.

- Allow handling of *Swift::Errors* even when the user disables the use of exceptions in C++.

Contribuições POC 1

- Classe genérica que representa uma exceção e a infraestrutura para "lançá-la" em Swift e "pegá-la" em C++.

- Classe genérica que representa um erro em Swift

Contributions

- The *Swift::Error* superclass in C++ to represent any Swift error.
- The infrastructure needed to generate a specific representation of a Swift error that inherits from the superclass defined above.
- The infrastructure needed to dynamically convert a *Swift::Error* to a subclass that correctly represents the thrown error case.
- The *Swift::Expected<T>* class that returns a *Swift::Error* or a value of type T.

Implementação: Primeira Parte

- Estender o gerador de código de C++ para reconhecer EnumDecls
- Implementar função para traduzir uma EnumDecl de Erro para uma subclasse de std::exception

Implementação: Segunda Parte

- Utilizando a Linguagem Intermediária do Swift
- Identificar que uma Enum do tipo Erro pode ser lançada
- Identificar qual caso será lançado

Resultado: Parte 1

```swift
public enum DivErrors : Error {
    case divByZero
}
```

Erro em Swift

```cpp
class DivErrors : public std::exception {
};

class divByZero : public DivErrors {
  virtual const char* what() const throw() {
    return "DivErrors.divByZero";
  }
} divByZero;
```

Exceção em C++

Resultado: Parte 2

```swift
public func divInt(num: Int, div: Int) throws -> Float {
    if (div != 0) {
        return Float(num / div);
    } else {
        throw DivErrors.divByZero
    }
}
```

Implementação da Função em Swift

Chamada da Função em C++

```cpp
inline float divInt(swift::Int num, swift::Int div) SWIFT_WARN_UNUSED_RESULT {
  void* opaqueError = nullptr;
  void* self = nullptr;
  float value = _impl::$s9Functions6divInt3num0B0SfSi_SitKF(num, div, self, &opaqueError);
  if (opaqueError != nullptr)
    throw divByZero;

  return value;
}
```

Accepted Contributions

1 Open ✓ 7 Closed | Author | Label | Projects | Milestones | Reviews | Assignee | Sort

- [Interop] [SwiftToCxx] Modify the swift::Error Dynamic Cast to return a Swift::Optional ✓ c++ interop — 7
  #62197 by Robertorosmaninho was merged 14 days ago • Approved
- [Interop] [SwiftToCxx] Introduce Dynamic Cast to swift::Error ✓ c++ interop — 13
  #61626 by Robertorosmaninho was merged on Oct 28
- [SwiftToCxx] Including Cxx representation of Swift's Error ✓ c++ interop — 7
  #60858 by Robertorosmaninho was merged on Sep 7 • Approved
- [Interop] [SwiftToCxx] Implementing a naive exception to be thrown in C++ if an Error is thrown in Swift ✓ c++ interop — 5
  #60079 by Robertorosmaninho was merged on Jul 20 • Approved
- [Interop] [SwiftToCxx] Using the hardcode noexcept flags on the compiler only to non throwing functions ✓ c++ interop — 24
  #59787 by Robertorosmaninho was merged on Jul 15 • Approved
- [Interop] [SwiftToCxx] Using the hardcode noexcept flags on the compiler only to non throwing functions ● c++ interop — 12
  #59259 by Robertorosmaninho was closed on Jun 29
- [Interop] [SwiftToCxx] New Throw Error Test ✓ c++ interop — 2
  #59217 by Robertorosmaninho was merged on Jun 2 • Approved

1 Open | 7 Closed | Author | Label | Projects | Milestones | Reviews | Assignee | Sort

- [Interop][SwiftToCxx] Introduces swift::Expected ✕ c++ interop — 24
  #61823 opened on Oct 31 by Robertorosmaninho

Example: Handling Errors in Division

Division.swift

```swift
@_expose(Cxx)
public enum DivByZero : Error {
    case divisorIsZero
    case bothAreZero

    public func getMessage() {
        print(self)
    }
}

@_expose(Cxx)
public func division(_ a: Int, _ b: Int) throws -> Float {
    if a == 0 && b == 0 {
        throw DivByZero.bothAreZero
    } else if b == 0 {
        throw DivByZero.divisorIsZero
    } else {
        return Float(a / b)
    }
}
```

snappify.com

Example: Handling Errors in Division

```
#include <cassert>
#include <cstdio>
#include "functions.h"

int main() {
    printf("Good example:\n");
    try {
        float result = Functions::division(4,2);
        printf("result = %f\n", result);
    } catch (Swift::Error& e) {
        auto errorOpt = e.as<Functions::DivByZero>();
        assert(errorOpt.isSome());

        auto errorVal = errorOpt.get();
        errorVal.getMessage();
    }
    return 0;
}
```

snappify.com

```
>./divisionException
Good example:
result = 2.000000
```

snappify.com

```
#include <cassert>
#include <cstdio>
#include "functions.h"

int main() {
    printf("Good example:\n");
    auto result = Functions::division(4,2);
    if (result.has_value()) {
        printf("result = %f\n", result.value());
    } else {
        auto optionalError =
            result.error().as<Functions::DivByZero>();
        assert(optionalError.isSome());

        auto errorValue = optionalError.get();
        errorValue.getMessage();
    }
    return 0;
}
```

snappify.com

Example: Handling Errors in Division with Exceptions

Division.cpp

```cpp
#include <cassert>
#include <cstdio>
#include "functions.h"

int main() {
    printf("Exception example:\n");

    try {
        auto result = Functions::division(1,0);
        printf("result = %f\n", result);
    } catch (Swift::Error& e) {
        auto errorOpt = e.as<Functions::DivByZero>();
        assert(errorOpt.isSome());

        auto errorVal = errorOpt.get();
        assert(errorVal == Functions::DivByZero::divisorIsZero);
        errorVal.getMessage();
    }
...
```

snappify.com

Division.cpp

```cpp
...
try {
        auto result = Functions::division(0,0);
        printf("result = %f\n", result);
    } catch (Swift::Error& e) {
        auto errorOpt = e.as<Functions::DivByZero>();
        assert(errorOpt.isSome());

        auto errorVal = errorOpt.get();
        assert(errorVal == Functions::DivByZero::bothAreZero);
        errorVal.getMessage();
    }
```

snappify.com

```
>./divisionException
Exception example:
divisorIsZero
bothAreZero
```

snappify.com

Example: Handling Errors in Division with Swift::Expected

DivisionExpected.cpp

```cpp
#include <cassert>
#include <cstdio>
#include "functions_expected.h"

int main() {
    printf("Runnnig expected example:\n");

    auto result0 = Functions::division(1,0);
    if (result0.has_value()) {
        printf("result = %f\n", result0.value());
    } else {
        auto optionalError = result0.error().as<Functions::DivByZero>();
        assert(optionalError.isSome());

        auto errorValue = optionalError.get();
        assert(errorValue == Functions::DivByZero::divisorIsZero);
        errorValue.getMessage();
    }
...
```

snappify.com

DivisionExpected.cpp

```cpp
...
auto result = Functions::division(0,0);
    if (result1.has_value()) {
        printf("result = %f\n", result1.value());
    } else {
        auto optionalError = result1.error().as<Functions::DivByZero>();
        assert(optionalError.isSome());

        auto errorValue = optionalError.get();
        assert(errorValue == Functions::DivByZero::bothAreZero);
        errorValue.getMessage();
    }
```

snappify.com

```
>./expectedExample
Runnnig expected example:
divisorIsZero
bothAreZero
```

snappify.com

Bridge header functions.h

```
inline Swift::ThrowingR
  void* opaqueError = nu
  void* _ctx = nullptr;
  auto returnValue = _i
  if (opaqueError ≠ nu
#ifdef __cpp_exceptions
    throw (Swift::Error
#else
    return SWIFT_RETURN
#endif

  return SWIFT_RETURN_T
}
```

```
#ifdef __cpp_exceptions

template<class T>
using ThrowingResult = T;

#define SWIFT_RETURN_THUNK(T, v) v

#else

template<class T>
using ThrowingResult = Swift::Expected<T>;

#define SWIFT_RETURN_THUNK(T, v) Swift::Expected<T>(v)

#endif
```



```
FT_WARN_UNUSED_RESULT {

aqueError);
```

Próximos Passos: POC 2

- Submeter as modificações como Pull Request para o repositório do Swift
- Criar mais casos de teste para validação da implementação atual
- Estudar e Implementar uma versão da proposta std::expected

References

Albatross. *History of C++. 2021*. Last accessed 6 May 2022.

Botet, J. B. V. *p0323r4 std::expected. 2017*. Last accessed 5 May 2022. Available on: <http://wg21.link/P0323r4>.

cplusplus.com. *Exceptions - C++ Tutorials. 2021*. Last accessed 6 May 2022. Available on: <https://www.cplusplus.com/doc/tutorial/exceptions/>.

Swift.org. *The Basics - The Swift Programming Language (Swift 5.6). 2022*. Last accessed 6 May 2022. Available on: <https://docs.swift.org/swift-book/LanguageGuide/TheBasics. html#ID335>.

Swift.org. *Swift.org - Welcome to Swift.org. 2022*. Last accessed 5 May 2022. Available on: <https://www.swift.org>.

Thanks!

Any questions? Email me!

robertogomes@dcc.ufmg.br