PROGRAMMIEREN II

DHBW Stuttgart Campus Horb INF2017

GENERIERUNG EINES MINESWEEPER FELDS

- Generiere ein Feld mit 16x16 Feldgröße und 99 Minen

- Fülle die Felder drumherum aus, basierend darauf wie viele Mienen um Sie herum sind

- Gib das Ergebnis am Ende aus

- Bonus: Implementiere die Spiellogik :-)

AGENDA

- Exceptions
- Function Templates
- Programmier-Aufgabe

EXCEPTIONS

- Zu deutsch: Ausnahme
- Bietet eine Möglichkeit den Programmfluss durch Kontrollsprünge zu verändern
- Die Kontrolle wird an Funktionen gegeben, die man Handler nennt

EXCEPTION BEISPIEL

```cpp
#include <iostream>

double divide(double a, double b)
{
    if (b == 0)
    {
        throw - 1;
    }
    return a / b;
}

int main()
{
    try
    {
        auto x = divide(4, 2);
        printf("Divided %f through %f result was: %f\n", 4.0, 2.0, x);
        auto y = divide(69.69, 0);
        printf("I am not sure if this gets printed\n");
        return 0;
    }
    catch (int ex)
    {
        printf("Caught exception!\n");
        return ex;
    }
}
```

EXCEPTION BEISPIEL ERKENNTNISSE

- Schlüsselwort try gibt den überwachten Block an
- Schlüsselwort throw wirft eine entsprechende Ausnahme
- Schlüsselwort catch leitet eine Handler-Funktion ein
- Exceptions führen zu einem geregeltem Stack Unwinding
- Das schlimmste: Jeder Typ kann als Exception geworfen / gefangen werden

DAS ÜBEL!

```
throw -1;
```

DIE LÖSUNG

- Innerhalb der C++ Standardlib erben alle geworfenen Exceptions von std::exception

- Diese Regel sollte man innerhalb seines Programms auch befolgen

- Überschreiben von exception::what , um die Fehlerursache anzugeben

CUSTOM EXCEPTION

```cpp
#include <iostream>
#include <exception>
using namespace std;
class too_young_exception : public exception
{
    virtual const char *what() const throw() override
    {
        return "The person you wanted to date is not an adult!";
    }
};
int main()
{
    try
    {
        printf("How old are you? - %i\n", 16);
        too_young_exception ex;
        throw ex;
    }
    catch (exception &e)
    {
        cout << e.what() << '\n';
    }
    return 0;
}
```

EXCEPTIONS DER STANDARDLIB

- std::bad_alloc
- std::bad_cast
- std::bad_exception
- std::typeid
- std::bad_function_call
- std::bad_weak_ptr

FÜR DIE VERERBUNG VORGESEHENE TYPEN

- std:: logic_error, wenn man ein interner Logikfehler festgestellt wird

- std::runtime_error, wenn man einen Fehler zur Laufzeit feststellt, der eher technisch ist (ungültige Pointer, fehlgeschlagene Lesevorgänge, …)

REIHENFOLGE VON CATCH BLÖCKEN

- Die Reihenfolge von Catch-Blöcken ist sehr wichtig

- Der erste Catch-Block der zutrifft wird ausgelöst, alle anderen werden ignoriert

- Folglich muss der „Elipsen“-Catch-Block (…) der letzte in der Reihe sein

MÖGLICHKEITEN FÜR EINEN CATCH-HANDLER

- Genau passender Typ
- Typ der durch implizierte Konvertierung passt
- Referenz auf einen passenden Typen
- (Referenz auf) Klasse / Struktur die auf eine base-class passt
- Elipse (…) passt auf alles

MULTIPLE CATCH BLÖCKE

```
// ...
try
{
    // ...
}
catch( ... )
{
    // Handle exception here.
}
// Error: the next two handlers are never examined.
catch( const char * str )
{
    cout << "Caught exception: " << str << endl;
}
catch( CExcptClass E )
{
    // Handle CExcptClass exception here.
}
```

ZERO COST EXCEPTIONS

- Die meisten C++ Compiler implementieren ein Pattern, das dafür sorgt, dass ein try-Block ohne Performance-Verlust durchlaufen werden kann

- Damit ist ein Try{}catch{} block im OK-Fall schneller als der return Type check (if(error))

- Exception Fall dafür deutlich teurer —> Merken

THEMENWECHSEL!

- Template Funktionen

ÜBERLADENE FUNKTIONEN

```
// overloaded functions
#include <iostream>
using namespace std;

int sum (int a, int b)
{
  return a+b;
}

double sum (double a, double b)
{
  return a+b;
}

int main ()
{
  cout << sum (10,20) << '\n';
  cout << sum (1.0,1.5) << '\n';
  return 0;
}
```

ÜBERLADENE FUNKTIONEN ERKLÄRUNG

- Zwei identische Funktionen bis auf die Typ-Parameter
- Für diesen Fall hat C++ (Funktions-)Templates eingeführt

TEMPLATE FUNCTION

```
#include <iostream>
template <typename myType>
myType sum(myType a, myType b, myType c)
{
    return a + b + c;
}

int main()
{
    // template type is derived from input parameters
    auto result = sum(1, 5, 3);
    printf("Result: %i\n",result);
    auto double_result = sum(0.444,0.69,4213.22);
    printf("Result: %f \n",double_result);
}
```

TEMPLATE DEKLARATION

- Eingeleitet durch Keyword template
- Die Template Paremeter werden als typename oder class (gleichbedeutend) beschrieben
- Es können auch „normale“ Typen als Template Parameter verwendet werden
-

BEISPIEL TEMPLATE

```cpp
#include <fstream>
#include <exception>
template <typename typ1>
typ1 readFromStream(std::ifstream &stream, char *dest)
{
    if (stream.is_open())
    {
        stream.read(dest, sizeof(typ1));
        typ1 *ptr = reinterpret_cast<typ1 *>(dest);
        return std::move(*ptr);
    }
    else
    {
        std::runtime_error("Could not read from file");
    }
}
```

PROGRAMM-CODE

```
int main()
{
    char *dest = new char[1024];
    try
    {
        std::ifstream stream("charakter.d2s");
        auto header = readFromStream<long>(stream, dest);
        printf("The correct header is: AA55AA55. File starts with %X\n“,
header);
    }
    catch (std::exception &ex)
    {
        printf("An error occured while reading the file: %s", ex.what());
        return -1;
    }
}
```

DISKUSSIONS-RUNDE

- Auf was muss man bei einem Binär-Format achten?

NÄCHSTE AUFGABE

- Auslesen der folgenden Informationen aus dem Diablo 2 Charakter im Repository
- Das Dateiformat ist in der eingecheckten PDF beschrieben
- Finde die folgenden Informationen über den Helden hinaus und speichere sie in einer entsprechenden Struktur / Klasse
    - Name
    - Status (lebendig, hardcore)
    - Titel
    - Klasse
    - Level
- Tipp: Mit ifstream::ignore können bytes übersprungen werden