Übungen OOSWE/Progr. 2 (C++) – KE 1

Der C/C++-Coding Styleguide ist einzuhalten.

Folgende Einstellungen sind für Debug und Release (All Configurations) vorzunehmen:

Einstellung	**Wert**
Solution Platform	x86
Properties->Conf. Properties->C/C++->General->Warning Level	Level4 (/W4)
Properties->Conf. Properties->C/C++->General->Treat Warnings As Errors	Yes (/WX)
Properties->Conf. Properties->C/C++->General->SDL checks	Yes (/sdl)
Properties->Conf. Properties->C/C++->Code Generation->Basic Runtime Checks	Default
Properties->Conf. Properties->C/C++->Code Generation->Security Checks	Enable Security Checks (/GS)

Legen Sie sich eine Solution an, die alle Aufgaben als Projekte enthält.

Aufgabe 1:

1.1 Implementieren Sie ein Programm, welches überladene Funktionen mit dem Namen „vPrint“ enthält. Legen Sie sich hierzu ein neues Projekt (KE01_AG1) mit der folgenden Datei an:

- Main.cpp (enthält alle Funktionen)

Die überladenen Funktionen akzeptieren die jeweiligen Übergabeparameter (jeweils ein Parameter) und geben den Wert dieser dann auf der Konsole aus. Die folgenden Datentypen sollen jeweils an die überladenen Funktionen übergeben werden:

int32_t, f64_t, const int8_t*, int8_t*, Point (Legen Sie sich für f64_t und f32_t eigene Typedefs an).

```
struct Point
{
   int32_t s32X;
   int32_t s32Y;
   uint8_t au8VeryLongUnusedDescription[100];
}
```

Testen Sie die Funktionsüberladung durch Aufrufen der Funktionen.

1.2 Implementieren Sie eine überladene Funktion „Add“, die zwei Parameter akzeptiert und diese addiert. Das Ergebnis wird mittels return zurückgegeben. Die überladenen Funktionen müssen folgende Datentypen akzeptieren:

- int32_t, int32_t –> Rückgabewert ist vom Typ int32_t
- int32_t*, int32_t* –> Rückgabewert ist vom Typ int32_t
- f32_t, f32_t –> Rückgabewert ist vom Typ f32_t
- Point, Point -> Rückgabewert ist vom Typ Point

Hinweis: Nach der Deklaration einer Struktur ist in C++ der Name der Struktur schon ein Datentyp. Das Aufführen von „struct“ ist jetzt nicht mehr notwendig.

Testen Sie die Funktionsüberladung durch Aufrufen der Funktionen. Prüfen Sie in einem Fall die Zeiger auf 0, nutzen Sie hierfür den Ausdruck „nullptr“.
Geben Sie beim Testen die jeweiligen Werte mit std::cout auf der Konsole aus. Was ist der Vorteil von << , >> im Vergleich mit printf?

Übungen OOSWE/Progr. 2 (C++) – KE 1

1.3 Die Übergabe von Objekten der Struktur „Point“ an eine Funktion ist ineffizient. Warum? Lösen Sie das Problem indem Sie einen Zeiger anstelle des Objektes übergeben. Überladen Sie hierfür die Funktion erneut. Achten Sie auf die Handhabung und überprüfen Sie auf nullptr.

- Point*, Point* -> Rückgabewert ist vom Typ Point

1.4 Vereinfachen Sie die Übergabe großer Objekte per Zeiger, indem Sie Referenzen verwenden. Überladen Sie die Methode hierfür erneut und übergeben Sie zwei Referenzen.

- Point&, Point& -> Rückgabewert ist vom Typ Point

Was fällt beim Übersetzen auf? Beheben Sie dies durch Auskommentierung einer Add-Funktion.

1.5 Definieren Sie ein konstantes Point-Objekt mittels dem „const“ Schlüsselwort. Übergeben Sie das konstante Objekt per Referenz an die überladene Funktion. Was fällt Ihnen auf? Überlegen Sie was das Problem ist und finden Sie einen Weg das Objekt zu übergeben, ohne „const“ zu entfernen.

1.6 In manchen Situationen wird der Rückgabewert einer Funktion zur Statusübermittlung (Ok, Fehler, …) genutzt. In diesen Fällen kann das Ergebnis, z.B. über einen Übergabeparameter an die Funktion, an den Aufrufer zurückgegeben werden. Zeiger bieten sich an, allerdings ist die Handhabung (nullptr, *,&, ->) komplizierter. Überladen Sie die Methode Add(Point,Point) erneut, und geben Sie das Ergebnis über einen zusätzlichen, dritten Referenzparameter, zurück. Das Ergebnis steht nach Ausführung der Funktion im Übergabeparameter. Um zu zeigen, dass es sich um einen sogenannten Output-Parameter handelt, hängen Sie an den Namen das Präfix _out an.

- Point&, Point&, Point& -> Rückgabewert ist vom Typ void

Überprüfen Sie den Inhalt der übergebenen Variable, welche das Ergebnis enthält.

1.7 Vermutlich sind bereits andere SW Engineers auf die Idee gekommen eine Funktion „Add“ zu implementieren. Das Ergebnis sind Namenskonflikte, wenn Sie diesen Programmcode in Ihrem Projekt verwenden. Sichern Sie sich ab und schieben Sie Ihre Funktionen in Ihren eigenen namespace „MyFunc“. Dann sind Ihre Funktionen nur noch über den Namespace MyFunc::Add eindeutig zu identifizieren.

Hinweis:
Vermeiden Sie zukünftig nach Möglichkeit das Statement „using namespace xxx“. Einzig in der cpp-Datei kann so auf den Namespace der h-Datei verwiesen werden.

Übungen OOSWE/Progr. 2 (C++) – KE 1

Aufgabe 2:
2.1 Implementieren Sie ein Programm, welches Standardwerte bei Übergabeparametern verwendet, um den Funktionsaufruf zu erleichtern.
Legen Sie sich hierzu ein neues Projekt (KE01_AG2) mit der folgenden Datei an:

- Main.cpp (enthält alle Funktionen)

Gegeben sei folgende Enumeration und Funktion:

```
enum class FileAccessMode
{
   Read = 0,
   Write,
   ReadWrite
};

void vOpenFile(const uint8_t* cpu8File, FileAccessMode eFileAccessMode);
```

Deklarieren Sie die Funktion so, dass im Standardfall automatisch „FileAccessMode::ReadWrite“ als Modus gewählt wird, wenn der Modus nicht explizit an die Funktion übergeben wird, da dieser Modus in den meisten Fällen verwendet wird. Die Funktion muss nicht implementiert werden. Ein Funktionsrumpf mit dem Makro _CRT_UNUSED ist hier genügend. Das Makro wird verwendet, um Warnungen zu unterdrücken.

2.2 Implementieren Sie die Funktion

```
f32_t f32Add(f32_t f32Val1, f32_t f32Val2, f32_t f32Val2)
```

erneut (ohne Namespace) und fügen Sie einen dritten Parameter mit
Standardwert ein, sodass die Funktion entweder zwei oder drei Werte addieren kann. Die Funktion f64Add muss dabei **nicht überladen** werden. Aufrufe wären dann z.B.:

```
f32Add(f32Val1, f32Val2);
f32Add(f32Val1, f32Val2, f32Val3);
```

2.3 Zeichenketten in C werden definiert durch einen Zeiger auf den Beginn der Zeichenkette sowie einem Nullterminator am Ende. C++ hat einen neuen Datentyp std::string um Zeichenketten zu definieren.
Definieren Sie eine Zeichenkette als std::string String1 mit dem Inhalt „Hallo C++“ und geben Sie das std::string Objekt anschließend auf der Konsole mit std::cout aus.

Geben Sie die Länge der Zeichenkette von String1 auf der Konsole aus, indem Sie String1.length() verwenden.

Auch std::string arbeitet intern mit Zeigern und einem Nullterminator. Wie dies und String1.length() funktioniert, wird nach der Einführung von Klassen gezeigt.

Definieren Sie eine zweite Zeichenkette std::string String2. Lesen deren Wert mit std::cin ein und geben Sie den eingelesenen Wert wieder mit std::cout aus.

Fügen Sie anschließend String2 an String1 an. Die string-Klasse bietet u.a. neben dem Zuweisungsoperator auch den +Operator an. Auf strcpy und strcat kann jetzt verzichtet werden. Geben Sie das Ergebnis anschließend auf der Konsole aus.

Übungen OOSWE/Progr. 2 (C++) – KE 1

Aufgabe 3:

Funktionen für verschiedene Datentypen zu überladen ist nicht immer sinnvoll, besonders wenn die Funktion für unterschiedliche Typen genau dieselben Operationen ausführt. Legen Sie sich hierzu ein neues Projekt (KE01_AG3) mit der folgenden Datei an:

- Main.cpp (enthält alle Funktionen)

3.1 Implementieren Sie ein Funktionstemplate, welches den Inhalt zweier Variablen des **gleichen Typs** tauscht. Der Name dieser Templatefunktion soll MySwap lauten.

Verifizieren Sie dies anhand verschiedener Datentypen (auch struct Point) und Konsolenausgaben.

3.2 Implementieren Sie ein Funktionstemplate „Add“ welches zwei übergebene Parameter unterschiedlichen Typs addieren kann, solange der +Operator für diese Variablen zur Verfügung steht. Welchen Vorteil bietet das Funktionstemplate gegenüber dem Überladen der Funktionen?

Verwenden Sie als Übergabeparameter z.B. diese Parameterpaare.

- f32_t + f32_t
- int32_t + f32_t
- f32_t + f64_t
- uint8_t + f64_t
- …

Rückgabewert soll auch ein Typparameter sein. Dieser ist dann allerdings immer beim Aufruf hinter die Funktion zu schreiben.

```
f32Val2 = Add<f32_t>(s32Val1, s32Val2);
u8Val = Add<uint8_t>(f64Val1, s32Val2); // Haben Sie hier ein Compilerfehler?
```

Verifizieren Sie die Funktionsweise anhand verschiedener Datentypen und Konsolenausgaben.

3.3 Mit dem C++-Keyword auto kann der Datentyp einer Variablen vom Compiler automatisch abgeleitet werden. Testen Sie auto und finden Sie heraus, welchen Datentyp die Variable vom Compiler zugewiesen bekommt.

Definition einer auto-Variablen:

```
auto ax1 = 5;
auto ax2 = 5.5;
auto ax3 = 5.5F;

Point sPoint1;
auto as = sPoint1;
```

Funktionsrückgabe:

```
auto asum1 = Add<uint32_t>(s32Val1, s32Val2);
auto asum2 = Add<f32_t>(f32Val1, s32Val2);
auto asum3 = Add<f64_t>(s32Val1, f32Val2);
```

Zur Ausgabe des auto-Datentyps ist dieses Template sinnvoll:

```
template <typename T>
std::string get_type_name(T var)
{
   return typeid(var).name();
}
```