

THE

C++

PROGRAMMING LANGUAGE

CPP-01 - C++ Language Overview

CPVR Vertiefungsmodul BTI-7281
Urs Künzler (urs.kuenzler@bfh.ch)




Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

CPP-01 Table of Contents – C++ Language Overview		
01.01	Object-Oriented Concepts of C++	4
01.02	A Brief History of C++	8
01.03	“Hello World” - C++ Version	12
01.04	C++ versus Java	13
01.05	C/C++ Program Compilation and Linking	16
01.06	Pre-Processing	17
01.07	main() Function	19
01.08	Anweisungen (Program Statements)	21
01.09	Funktionen	23

CPP-01

2



Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

CPP-01

Table of Contents – C++ Language Overview II

01.10	Data Types	26
01.10.1	Vordefinierte Datentypen (Built-In Types)	27
01.10.2	Benutzerdefinierte Datentypen	29
01.10.3	Pointer and References	31
01.10.4	Arrays and C-Strings	38
01.10.5	Deklaration und Definition von Variablen	41
01.10.6	Geltungsbereich und Lebensdauer (Scope)	43

CPP-01

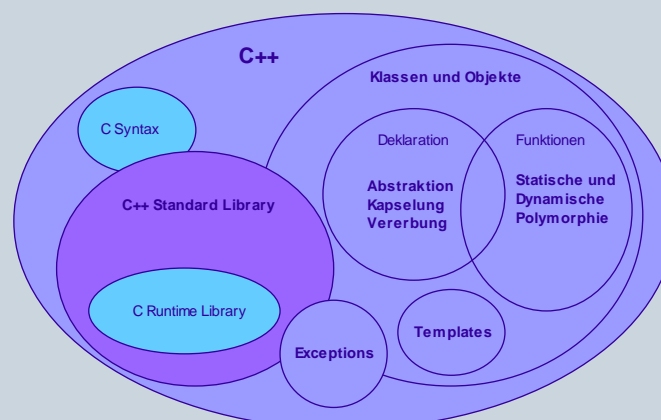
3

B Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

01.01

Object-Oriented Concepts of C++

C++



CPP-01

4

B Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Object-Oriented Concepts of C++ II



- **Abstraktion**
 - Bildung von **Klassen** als benutzerdefinierter abstrakter Datentyp
 - Eine Klasse setzt sich zusammen aus:
 - **Data Members** (Synonyme: Attribute, Variablen, Properties)
 - **Function Members** (Synonyme: Operationen, Funktionen, Methoden)
 - **Instanziierung** erlaubt die Erzeugung von (Klassen-) **Objekten**
- **Kapselung**
 - Angabe der **Zugriffsrechte** für Funktionen und Daten
- **Modularisierung**
 - Aufteilung von Interface und Implementation einer Klasse auf separate Files
- **Typenprüfung**
 - Strikte Typenprüfung erweitert für Klassen (inklusive parametrisierte Klassen)

CPP-01 5

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Object-Oriented Concepts of C++ III



- **Hierarchie**
 - **Klassenhierarchie** zur Beschreibung hierarchischer Beziehungen zwischen Klassen
 - **Vererbung** der Eigenschaften einer Basisklasse (Funktionen und Daten) an die davon abgeleiteten Klassen
 - **Aggregation** erlaubt die Verschachtelung von Klassen (lokale Klassen)
- **Polymorphismus**
 - **Statische (signaturegebundene) Polymorphie**: Unterscheidung von Funktionen mit gleichem Namen aber unterschiedlicher Signatur (Function Overloading)
 - **Dynamische (vererbungsgebundene) Polymorphie**: Unterscheidung von Funktionen mit gleichem Namen und Signatur aber in voneinander abgeleiteten Klassen (Function Overriding)
- **Generizität**
 - Generische Datentypen durch **Template-Klassen** implementierbar

CPP-01 6

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Object-Oriented Concepts of C++ IV



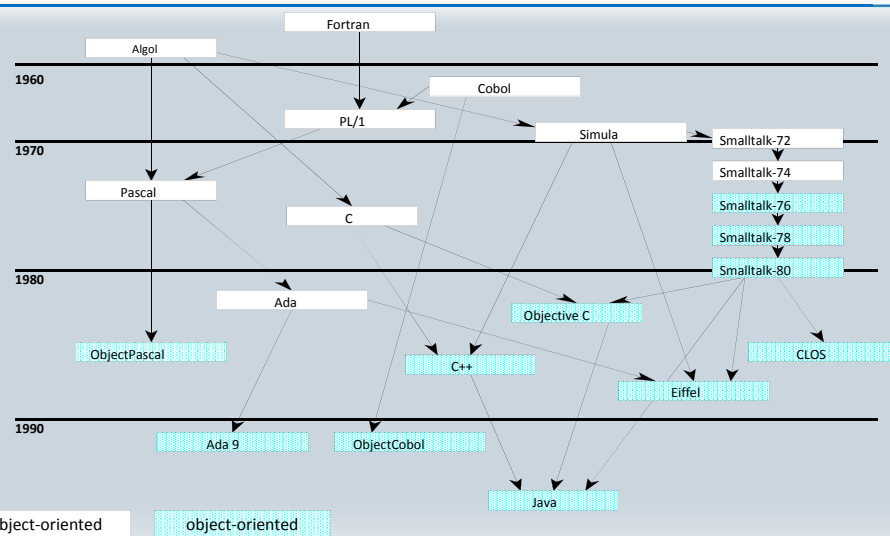
- **Exception Handling**
 - Mechanismus zur Behandlung von erwarteten und unerwarteten Laufzeitfehlern
- **Persistenz**
 - Durch C++ nicht direkt unterstützt (Verwendung einer Klassen-Bibliothek)
- **Gleichzeitigkeit**
 - Erst mit C++11 direkt unterstützt (Verwendung einer Klassen-Bibliothek z.Bsp. Boost)

CPP-01 7

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

01.02

A Brief History of C++



CPP-01 8

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

A Brief History of C++ II



- **1979:**
 - Beginn der Entwicklung durch Bjarne Stroustrup für AT&T internen Gebrauch ("C with Classes")
- **1983:**
 - Erster Gebrauch von C++ ausserhalb von AT&T
- **1985:**
 - Erstes kommerzielles C++ Entwicklungsprodukt
- **1989:**
 - Bildung eines Komitees zur Standardisierung von C++ innerhalb der ANSI (American National Standards Institute)



Bjarne Stroustrup
(The Creator of C++)

CPP-01

9

B Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

A Brief History of C++ III



- **1991:**
 - Definition des "C++ Standards" durch B. Stroustrup in "The C++ Programming Language, 2nd Edition"
- **1991:**
 - Zusammenführung des ANSI-Komitees und Gremien anderer Länder unter der ISO
- **1995:**
 - Veröffentlichung des "Draft C++ Standard"
- **1996:**
 - Praktischer Abschluss der Standardisierung
- **1998:**
 - Internationaler ANSI/ISO C++ Standard (ca. 770 Seiten)
- **2011:**
 - Aktualisierter ANSI/ISO Standard (11.10.2011) → C++11
- **2014**
 - Aktualisierter ANSI/ISO Standard mit kleinen Korrekturen zu C++11 (15.12.2014) → C++14

CPP-01

10

B Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

A Brief History of C++ IV



CPP-01

11

 Berner Fachhochschule
 Haute école spécialisée bernoise
 Bern University of Applied Sciences

01.03

"Hello World" - C++ Version



- ANSI C definiert ein Subset von C++
- C++ Syntax Definition als Erweiterung der C Syntax
- Beispiel - "Hello World":

```
#include <iostream>
#include <string>
using namespace std;

int main(void)
{
    string str;

    cout << "Please Enter Your Name: ";
    cin >> str;
    cout << "Hello, " << str << "!" << endl;
    return 0;
}
```

CPP-01

12

Demo: CPP-01-D.00_HelloWorld

 Berner Fachhochschule
 Haute école spécialisée bernoise
 Bern University of Applied Sciences

01.04

C++ versus Java



Why to use C++

- C/C++ compilers create native assembly code for the CPU the program runs on and can therefore directly be executed.
- C/C++ allows direct access to hardware and operating system APIs.
- Where as Java was developed for easy and safe programming, C++ offers more possibilities to fine tune the performance of programs.
- Due to the runtime performance, C/C++ is still the language of choice for applications in the field of interactive Computer Graphics, Image Processing, Virtual- and Augmented Reality.

CPP-01

13

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

C++ Features Not Available in Java



- Pre-processor
- Namespaces
- Pointers
- References
- Global Variables / Functions
- Default Arguments for function members
- Operator Overloading
- Inline Declaration
- Friend Members
- Multiple Inheritance
- Private Inheritance
- Pure Virtual Methods
- (Template Classes and Functions)

CPP-01

14

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Java Features Not Available in C++



- Reflection
- Serialization
- Interfaces
- Keywords: `super` / `final`
- Garbage Collection
- Wrapper classes for primitive types (e.g. `integer`)
- (Comprehensive class library for file formats, compression, etc.)

CPP-01 15

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

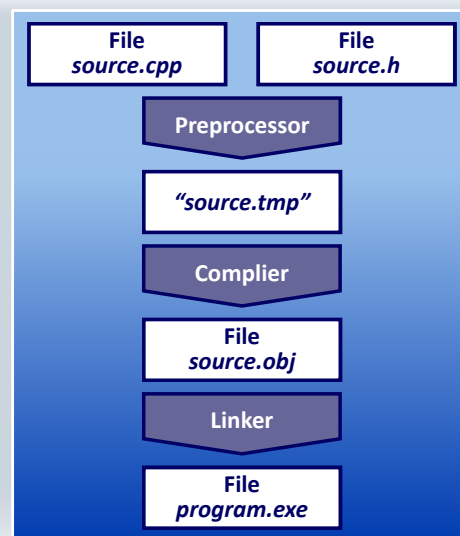
01.05

C/C++ Program Compilation and Linking



- C/C++ program build is a three step process:

1. Pre-processing of source files
2. Compilation to object files (single pass compiler)
3. Linking of object files creates program executable



CPP-01 16

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

01.06

Pre-Processing



- Die erste Phase der C++ Compilation besteht in der Verarbeitung durch den Preprocessor (# Directives) und liefert folgende Tokens:
 - Trennzeichen (Whitespace): space, tab, formfeed, newline (Windows vs. Unix!)
 - Kommentare (Comments) : /* multiline comment */ und // end of line comment
 - Bezeichner (Identifier): case-sensitive Variablenamen mit "beliebiger" Länge
 - Schlüsselwörter (Keywords): C erweitert um neue C++ Keywords (30 Stk.)
 - Literale (Literals): integer, float, character, string, boolean Konstanten
 - Operatoren (Operators): Vergleichsoperatoren (==, >=), Zuweisung (+=, *=), etc. C Operatoren erweitert um **new** und **delete** sowie dem Scope Operator (::)

CPP-01

17

 Berner Fachhochschule
 Haute école spécialisée bernoise
 Bern University of Applied Sciences

Pre-Processing Directives



- The Pre-processor is used to evaluate pre-processing directives:
 - **#include** insert the contents of another file
 - **#define** define variables
 - **#undef** used to undefine variables
 - **#error** display an error message
 - **#pragma** compiler specific commands
 - **#ifdef, #else, #endif** conditional operators

```
#ifdef DEBUG
cout << "This is the DEBUG version, i=" << i << endl;
#else
cout << "This is the RELEASE version!" << endl;
#endif
```

CPP-01

18

 Berner Fachhochschule
 Haute école spécialisée bernoise
 Bern University of Applied Sciences

01.07

main() Function



- **Einsprungpunkt eines C++ Programm bildet die Funktion main():**
 - main() ist eine globale Funktion.
 - main() muss in einem Programm genau einmal definiert werden.
 - main() darf programmintern nicht als Funktion aufgerufen werden.
 - main() darf nicht überladen werden.
 - main() darf nicht static oder inline definiert werden.
 - Die Adresse von main() kann programmintern nicht ermittelt werden.
- **Vor dem Aufruf von main(), werden in C++ die Konstruktoren von globalen und static Objekten aufgerufen (Static Initialization).**
- **Analog dazu werden in C++ nach dem Verlassen von main(), die Destruktoren dieser Objekte aufgerufen (Static Destruction).**

CPP-01

19

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

main() Function II



- Der C++ Standard definiert zwei Implementation von main():
 - `int main(void)`
 - `int main(int argc, char *argv[])`
- Weitere (implementationsabhängige) Parameter können vom Compiler-Hersteller hinzugefügt werden.
 - Beispiel (MS): `int main(int argc, char *argv[], char *envp[])`
- Der Rückgabewert von main() wird definitionsgemäss als Error-Code interpretiert und ans Betriebssystem zurückgegeben. (Rückgabewert = 0 entspricht fehlerfreiem Programmdurchlauf)
- Erfolgt in main() keine explizite return Anweisung, wird in C++ implizit ein "return 0;" ausgeführt.

CPP-01

20

Demo: CPP-01-D.01_MainArguments

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

01.08

Anweisungen (Program Statements)



- **Selektionen**

- if-else statement: Auswahl zwischen zwei Alternativen
- switch-case statement: Auswahl zwischen mehreren Alternativen

- **Iterationen**

- while statement: Schleife mit Prüfung einer Anfangsbedingung
- do-while statement: Schleife mit mindestens einer Durchführung
- for statement: Schleife mit einer genau definierten Anzahl Iterationen

- **Sprunganweisungen**

- break statement: Zum Verlassen einer switch oder Iterations-Anweisung
- continue statement: Springt ans Iterationsende und überprüft die Abbruchbedingung
- return statement: Zum Rücksprung aus einer Funktion evtl. mit Rückgabewert
- goto statement: Zum direkten und unbedingten Sprung auf ein Programm-Label

CPP-01

21

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Ausdrücke und Typenkonversion



- **Ein Ausdruck ist eine Verknüpfung von Literalen, Operatoren und Identifiern. Mehrere Ausdrücke werden durch Strichpunkt getrennt.**

- Beispiel: `a = 15.5 * (b + CalcValue(c));`

- **Die Auflösung von Ausdrücken erfolgt unter Berücksichtigung der Prioritäten und Auswertungsrichtung von Operatoren.**

- **Mit Klammern wird die Auswertungsreihenfolge explizit festgelegt.**

- **Werden in einem Ausdruck verschiedene Datentypen verwendet, ist eine implizite (durch den Compiler) oder explizite (durch den Programmierer) Typenkonversion notwendig.**

CPP-01

22

http://en.cppreference.com/w/cpp/language/operator_precedence

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

01.09

Funktionen



- Eine Funktion ist ein in sich abgeschlossenes Teilprogramm dem Parameter übergeben werden können und das einen Rückgabewert zurückgeben kann.
- Neben Keywords definiert der C++ Standard eine Anzahl von Standardklassen und -funktionen, welche den Funktionsumfang und die Anwendbarkeit von C++ erheblich vergrößern. (Diese sind in der C++ Standard-Klassenbibliothek zusammengefasst.)
- Funktionen können rekursiv verschachtelt werden.

CPP-01

23

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Aufbau und Verwendung von Funktionen



- Bevor eine Funktion verwendet werden kann muss diese im Kontext definiert oder zumindest deklariert werden. Die Predeklaration erfolgt meist in einem Header-File.
 - Beispiel: `void ShowMessage(char*, int, int);`
- **Ablauf eines Funktionsaufrufs:**
 - Beim Aufruf der Funktion wird ein Stack (Automatic Memory) erzeugt in welchem die Funktionsargumente sowie lokale Variablen abgelegt werden.
 - Der Funktionsname wird als Einsprungpunkt für die Code Ausführung verwendet.
 - Beim Verlassen der Funktion wird ein allfälliger Rückgabewert mit `return` zurückgegeben und der Stack (inklusive lokaler Variablen) gelöscht.

CPP-01

24

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Syntax von Funktionen



■ Syntax einer Funktionsdefinition:

```
[Spezifizierer][Rückgabety] Identifier ([Typ Arg1], ...) {  
    Statements;  
    [return Rückgabewert;]  
}
```

■ Spezifizierer:

- **inline** - Compiler-Hinweis den Funktionscode direkt einzufügen (nicht aufzurufen).
- **virtual** - Erlaubt Member-Funktionen in einer abgeleiteten Klasse neu zu definieren.

■ Rückgabety:

- Als Rückgabety können mit Ausnahme von Array und Funktionen beliebige Daten-typen, insbesondere auch Objekte, verwendet werden. (Array und Funktionen Rückgabewerte können aber mit einem Pointer realisiert werden).
- Die Werterückgabe erfolgt mit dem **return** statement.

CPP-01

25

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

01.10

Data Types



■ Built-In Data Types

■ Custom Data Types

■ Derived Data Types

■ Scope of Variables

CPP-01

26

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

01.10.1

Vordefinierte Datentypen (Built-In Types)



- Vordefinierte Datentypen sind integraler Bestandteil der Sprachdefinition und dienen als Basis für alle anderen Typen.
- C++ kennt folgende vordefinierten Datentypen:
 - Ganzzahlige Datentypen: `char, short, int, long` (vgl. `<limits.h>`)
jewels mit `signed` oder `unsigned` Präfix
 - Gleitkomma-Datentypen: `float, double, long double` (vgl. `<float.h>`) min Epsilon ist dort definiert
 - Void Datentyp: `void` oder `void*` (Pointer auf unbekannten Typ)
 - Multibyte Zeichen Datentyp: `wchar_t` (wide char type) für `utf-16, utf32`
 - Boolescher Datentyp: `bool` Datentyp mit Wertebereich `true` oder `false`
- Der Wertebereich von Ganzzahl- und Gleitkomma-Datentypen ist Compiler und Plattform abhängig!

CPP-01

27

Demo: `<limits.h>` und `<float.h>`

 Berner Fachhochschule
 Haute école spécialisée bernoise
 Bern University of Applied Sciences

Boolescher Datentyp (Built-In Type)



- Mit ANSI/ISO C++ eingeführter, vordefinierter logischer Datentyp.
- Ein Boolescher Datentyp wird durch das Keyword `bool` definiert und kann die Literale `true` oder `false` als Werte annehmen.
- Datentyp `bool` ist mit `int` kompatibel (implizite Typkonversion):
 - `(int) true` - entspricht gemäss Definition 1
 - `(int) false` - entspricht gemäss Definition 0
 - `(bool) i` - ist `false` für `i=0` und `true` für alle `i!=0`
 - Null-Pointer werden implizit zu `false`, alle anderen zu `true` konvertiert

▪ Beispiele:

```
bool b1 = (a == b); // b1 ist true falls a gleich b ist
bool b2 = 7;        // b2 wird mit true initialisiert
int i = true;        // i wird mit 1 initialisiert
```

CPP-01

28

 Berner Fachhochschule
 Haute école spécialisée bernoise
 Bern University of Applied Sciences

01.10.2 Benutzerdefinierte Datentypen



- Mit benutzerdefinierte Datentypen können aus vordefinierten Datentypen neue abstrakte Datentypen definiert werden
- C++ kennt folgende benutzerdefinierten Datentypen:
 - Eigener Typ: `typedef unsigned char BYTE;`
 - Aufzählung: `enum Color {red, green, blue};`
 - Union: `union Value {int ival; double dval; char* cval;};`
 - Struktur: `struct Date {int day; int month; int year;};`
 - Klasse: `class Point {int xCoord; int yCoord;};`

CPP-01 29

B Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Abgeleitete Datentypen



- Aus den vordefinierten Basisdatentypen lassen sich mittels Operatoren die folgenden weiteren Datentypen ableiten:
 - Feld (Array): `type array_name[number];`
 - Zeichenkette (String): `char string_name[number];` (char-Array mit '\0' Terminierung)
`char myText[] = "This is my text"; // implizites '\0'`
 - Referenz (Reference): `type& reference_name;`
 - Zeiger (Pointer) : `type* pointer_name; (oder type *pointer_name;)`

Initialisierung mittels 'NULL' oder '0' (Null-Pointer), wobei das C++ Runtime-System gewährleistet, dass sich an der Adresse 0x00000000 kein gültiges Objekt befindet.

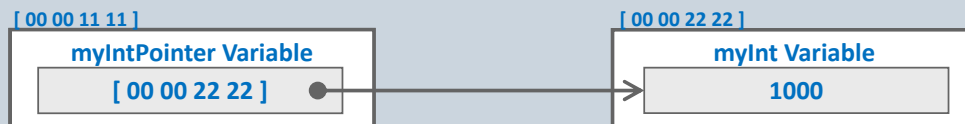
CPP-01 30

B Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

01.10.3 Pointer and References



- Whereas Java uses pointers only internally, C++ uses pointers explicitly and programmers need to decide where to use them.
- A pointer is a variable that stores the address to another memory location. The size of a pointer can be determined at compile time.



- Example:**

```

int myInt = 1000;           // int definition
int* myIntPtr = 0;         // int pointer definition
myIntPtr = &myInt;         // address assignment
cout << " Int:" << *myIntPtr; // output int value (1000)
cout << " Addr:" << myIntPtr; // output int address (00002222)
cout << " pAddr:" << &myIntPtr; // output pointer addr. (00001111)
  
```

CPP-01

31

Pointers



- Pointers are used for the following reasons:**
 - A single pointer variable can point to different values during program execution.
 - A pointer variable will point to a single value, but it is not known at compile time.
 - Pointers are required if the amount of memory to allocate can not be determined at compile time but needs to be dynamically allocated at run time.
- Pointers can (should) be initialized with NULL or 0 (i.e. null pointer)**
 - C++ runtime system guarantees that no valid object can be at memory address 0x0000
- Valid operations with pointers are:**
 - Pointers can be tested for equality with other pointers or with the null pointer
 - Pointers can be assigned with other pointers or with the address operator &
 - Pointers can be dereferenced with operator * (i.e. provide access to type pointed to)
 - Pointers can be dereferenced and provide member access with the pointer operator ->
 - Pointers used for arrays can be subscripted with the index operator []
 - Array pointers can be used for pointer arithmetic by adding or subtracting integers

CPP-01

32

Demo: CPP-01-D.02_Pointers

References



- Eine Referenz beschreibt einen alternativen Namen (Alias) für eine bestehende Variable, Konstante oder Funktion.
- Referenzen müssen bei ihrer Definition initialisiert werden.
- Eine Referenz wird implizit immer als konstant definiert. Der referenzierte Name einer Referenz kann deshalb nach der Initialisierung nicht mehr geändert werden.
- Referenzen können als konstante Pointer verstanden werden, die bei jeder Verwendung automatisch de-referenziert werden.

CPP-01 34

B Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

References II



- Beispiele - Definition und Verwendung von Referenzen:

```
int x;           // Deklaration der Variablen x
int* p;         // Deklaration eines Pointer p auf int
int i = 1;      // Definition und Initialisierung der Variablen i
int& r = i;     // Definition einer Referenz r auf i
int& r1;        // Compiler Fehler: "references must be initialized"

...

x = r;          // x = 1

r = 2;          // i = 2, r = 2, x bleibt 1 da x keine Referenz ist
i++;           // i = 3, r = 3
r++;           // i = 4, r = 4

p = &r;         // Zuweisung der Adresse von r an p (&r == &i)
(*p)++;        // i = 5, r = 5
```

CPP-01 35

Demo: CPP-01-D.03_References

B Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Call by Value versus Call by Reference



- **Argumente:**
 - Die Übergabe der Argumente erfolgt durch *call by value* oder *call by reference*.
- **Call by Value:**
 - Ein *Kopie* des Parameters wird auf dem Stack übergeben (sehr aufwändig für Objekte). Änderungen des Parameters bleiben lokal. via copy konstruktor aufruf
- **Call by Reference:**
 - Eine *Reference* oder ein *Pointer* auf einen Parameter wird auf dem Stack übergeben (kleiner Aufwand, immer 4 bzw. 8 Byte). Lokale Änderungen des Parameters werden auch im aufrufenden Code wirksam.
 - Ein "Call By Reference" Aufruf kann nur verwendet werden, wenn lokale Änderungen am Parameter an die aufrufende Funktion zurückgegeben werden sollen.

CPP-01 36

B Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Call by Value versus Call by Reference II



- **Beispiel - Referenzen als Funktionsparameter:**

```
// Implementation mit Pointer
void Increment( int* iValue ) { (*iValue)++; }

void MyFunction() {
    int iLocalValue = 1;
    Increment( &iLocalValue );          // iLocalValue = 2
}

// Implementation mit Referenz
void Increment( int& iValue ) { iValue++; }

void MyFunction() {
    int iLocalValue = 1;
    Increment( iLocalValue );           // iLocalValue = 2
}    increment(&iLocalValue) --> für pointer implementation
```

CPP-01 37

Demo: CPP-01-D.04_FunctionCall

B Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

01.10.4 Arrays and C-Strings



- An array is a homogenous list of elements that are stored in consecutive memory locations.
- **Syntax:**
 - <base type> <identifier> [<size>]; Example: `double DoubleArray[100];`
- Arrays are 0-based, that is the first item in an array is at position 0.
- By means of the index operator [] each element of an array can be accessed directly (and randomly) by its position index in the array.
- An array variable in C/C++ is actually the address of the first element. That is, an array is a constant pointer to the base element of an array.

CPP-01

38

Demo: CPP-01-D.05_Arrays



C-Strings - Null-Terminated Character Arrays



- In C/C++ a (C-)String is a null-terminated character array. That is, an array of char of which the end of the data is marked by a zero value. The symbol for this is `'\0'` (ASCII NUL char).
- A string delimited by double quotes is automatically terminated by `'\0'`
- **Example:**

```
char myText[] = "my text"; // ends with implicit '\0'
int myInts[] = {5,6,7,8}; // explicit int initialization
```
- C-Strings are rather difficult to use and are a constant source for errors (i.e. buffer overruns due to limited buffer size).
- The standard library of C++ contains a string class, which can be accessed through the `<string>` include file and offers the usual functions of a string class with dynamic buffer size adjustment.

CPP-01

39

Demo: CPP-01-D.05_Arrays



include einschub
 z.b: string.h -> c-string include
 string -> cpp string klasse
 cstring string.h im std namespace

01.10.5

Deklaration und Definition von Variablen



- Jede Variable ist vor ihrer Verwendung **genau einmal** zu definieren.
 - Notation: `[storage class] type variable_name [= literal];`
- Bei der **Deklaration** einer Variablen wird ein eindeutiger Identifier deklariert und mit einem Datentyp verknüpft, **ohne Speicher zu allozieren** (kann mit **extern** Keyword auch mehrmals erfolgen). Beispiel: `int i;`
- Bei der **Definition** einer Variablen wird zusätzlich der dazugehörige Speicher alloziert (genau einmal). Beispiel: `int i = 10;`
- In C++ muss die Definition einer Variable nicht am Anfang eines Blockes stehen, sondern kann an einer beliebigen Stelle erfolgen.
 - Beispiel: `for(int i=0; i<max; i++) { cout << i; ... }`
- (Lokale) Variablen sollten bei der Definition initialisiert werden.

CPP-01

41

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

const Variablen



- Bei Definition einer Variablen mit **const** kann diese nachträglich nicht verändert werden (Symbolische Konstante) und muss deshalb zwingend initialisiert werden.
 - Beispiel: `const double PI = 3.14159;`
- Definition symbolischer Konstanten mittels **#define** Anweisung ist veraltet und schwierig im Unterhalt (z.B. Debugging).

CPP-01

42

 Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

01.10.6

Geltungsbereich und Lebensdauer (Scope)



- Der Programmbereich innerhalb dem eine Variablendefinition gültig ist, wird als Geltungsbereich (Scope) bezeichnet.
- Meist bestimmt der Scope auch die Lebensdauer einer Variablen.
- Beim Geltungsbereich von Variablen wird unterschieden zwischen:
 - Global Scope: Für (globale) Variablen, definiert ausserhalb eines Blockes
 - Local Scope: Für (lokale) Variablen, definiert innerhalb eines Blockes
- Der Geltungsbereich einer globalen Variablen beginnt am Definitionspunkt und erstreckt sich bis zum Ende des C++ - Files.
- Der Geltungsbereich einer lokalen Variablen beginnt am Definitionspunkt und erstreckt sich bis zum Ende des Blockes welcher die Variablen-Definition umgibt.

CPP-01

43

 Berner Fachhochschule
 Haute école spécialisée bernoise
 Bern University of Applied Sciences

Scope (Resolution) Operator



- Der Scope Operator (::) wird für die explizite Angabe des Scopes eines Identifiers verwendet (Problem der Namensverdeckung).
- Beispiel - Qualifizierung mittels Global Scope Operator:

```

int n = 0;                                // globale Definition

int main(void)
{
    cout << "n=" << n << endl;           // Ausgabe: n=0

    int n = 1;                             // lokale Definition 1
    cout << "n=" << n << endl;           // Ausgabe: n=1
    {
        int n = 2;                         // lokale Definition 2
        cout << "n=" << n << endl;       // Ausgabe: n=2
    }
    cout << "n=" << n << endl;           // Ausgabe: n=1
    cout << "n=" << ::n << endl;         // Ausgabe: n=0 (Global)

    return 0;
}

```

CPP-01

44

Demo: CPP-01-D.06_ScopeOperator

 Berner Fachhochschule
 Haute école spécialisée bernoise
 Bern University of Applied Sciences