University of Applied Sciences
HOCHSCHULE EMDEN·LEER Networked Systems

Programmieren 3
C++

Vorlesung 03
Container, Strings, Klassen

Prof. Dr. Dirk Kutscher
Dr. Olaf Bergmann

Wiederholung

Heute

- Basics, die man benötigt, um erste praktische Schritte zu machen
  - Standardbibliothek
  - IO
  - Container
  - Strings
  - Pointer vs. Referenzen

Standardbibliothek

- Tools & Mechanismen zur Interaktion mit Betriebssystem
  - Ein-/Ausgabe
  - Strings
  - Container (Listen, Vektoren usw.)
  - … und vieles mehr
- Wie bei C (`stdio.h` usw.) — nur viel umfangreicher

C++ Standard Library

- Input/output
- Strings

Standard Template Library

- `algorithm`
- `functional`
- Sequence containers
- Associative containers
- Unordered associative containers

C standard library

- Data types
- Character classification
- Strings
- Mathematics
- File input/output
- Date/time
- Localization
- Memory allocation
- Process control
- Signals
- Alternative tokens

Miscellaneous headers:

- `<assert.h>`
- `<errno.h>`
- `<setjmp.h>`
- `<stdarg.h>`

V•T•E

Definitionen vs. Deklarationen

Deklaration

```
extern int a;                                   // Declaring a variable a without defining it
struct _tagExample { int a; int b; };          // Declaring a struct
int myFunc (int a, int b);                      // Declaring a function
```

```
int a;
int b = 0;
int myFunc (int a, int b) { return a + b; }
struct _tagExample example;
```

Definition

Lädt die in iostream enthaltenen Deklarationen

```
// my first program in C++
#include <iostream>

int main()
{
  std::cout << "Hello World!";
}
```

Eingabe

Wir verwenden alle Deklarationen aus dem Namespace `std` ohne Präfix

```
// i/o example

#include <iostream>
using namespace std;

int main ()
{
  int i;
  cout << "Please enter an integer value: ";
  cin >> i;
  cout << "The value you entered is " << i;
  cout << " and its double is " << i*2 << ".\n";
  return 0;
}
```

Objekt für die Default-Eingabe (z.B. Konsole)

Eingabe-Operator

Dateien

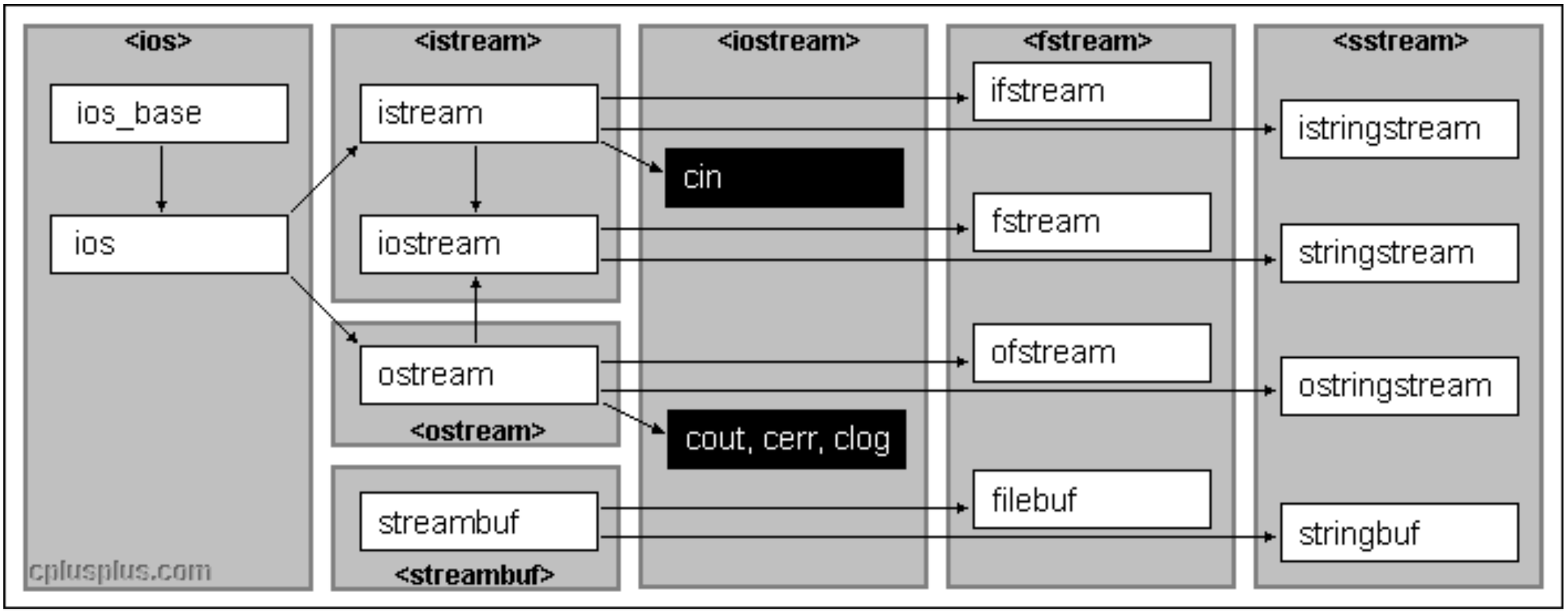
Klasse für schreibbare Dateien

Neues Objekt — lokale Variable

```
#include <fstream>
int main()
{
    std::ofstream file("file.txt");
    file << "Hello, world!" << std::endl;
}
```

Variable, die `Zeilenende` repräsentiert

iostream-Klassen



Vectors in C++

Container-Typ `vector`

Element-Typ `int`

Variablenname

Initiale Größe: 5 Elemente

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
  vector<int> values(5);

  cout << "Enter 5 integers: ";
  // taking input and storing it in a vector
  for(int i(0); i<5; ++i) {
    cin >> values[i];
  }
  cout << "Displaying integers: ";
  // printing elements of an array
  for(int i(0); i<5; ++i) {
    cout << values[i] << endl;
  }
  return 0;
}
```

Arrays in C vs. Vectors in C++

C

```c
int values[5];
```

```c
scanf("%d", &values[i]);
```

```c
for(int i = 0; i < 5; ++i) {
     copyOfValues[i]=values[i];
}
```

C++

```c
vector<int> values(5);
```

```c
cin >> values[i];
```

```c
copyOfValues=values;
```

Heute:

Container, Strings und Klassen

Container in C++ (Stdlib)

Sequence containers

Sequence containers implement data structures which can be accessed sequentially.

array (C++11)	static contiguous array (class template)
vector	dynamic contiguous array (class template)
deque	double-ended queue (class template)
forward_list (C++11)	singly-linked list (class template)
list	doubly-linked list (class template)

Associative containers

Associative containers implement sorted data structures that can be quickly searched ($O(\log n)$ complexity).

set	collection of unique keys, sorted by keys (class template)
map	collection of key-value pairs, sorted by keys, keys are unique (class template)
multiset	collection of keys, sorted by keys (class template)
multimap	collection of key-value pairs, sorted by keys (class template)

https://en.cppreference.com/w/cpp/container

Zeichenketten in C

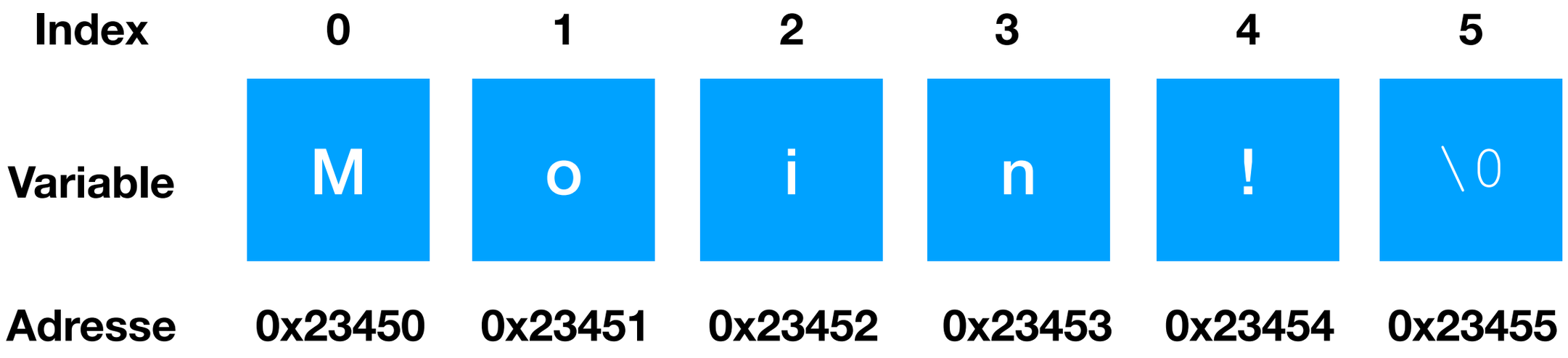
- In C: ein-dimensionale Arrays aus Buchstaben mit einem Null-Byte am Ende.

```c
#include <stdio.h>

int main () {
  const char greeting[]="Moin!";
  printf("Greeting message: %s\n", greeting);
  return 0;
}
```

Zeichenketten in C

- In C: ein-dimensionale Arrays aus Buchstaben mit einem Null-Byte am Ende.

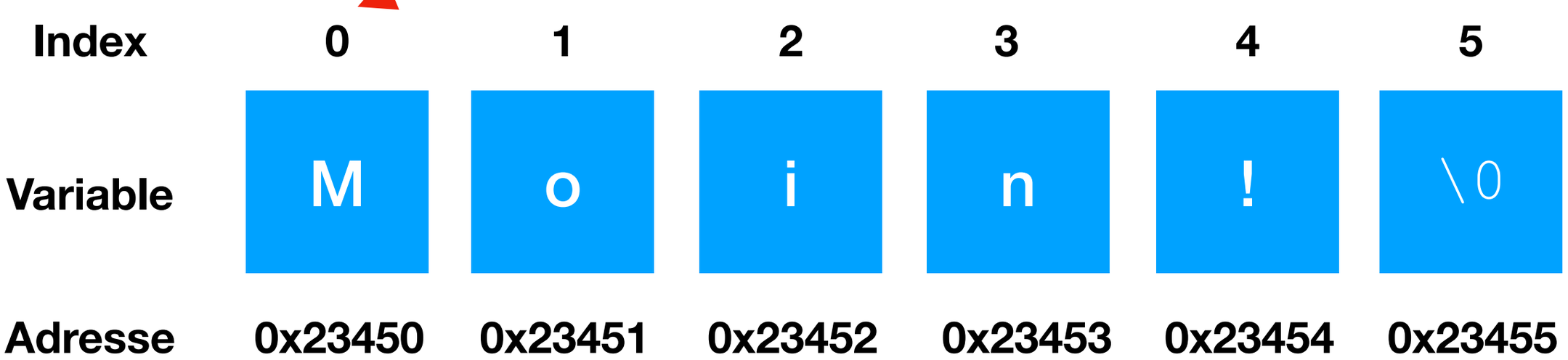
```c
#include <stdio.h>

int main () {
  const char greeting[]="Moin!";
  printf("Greeting message: %s\n", greeting);
  return 0;
}
```



Zeichenketten in C

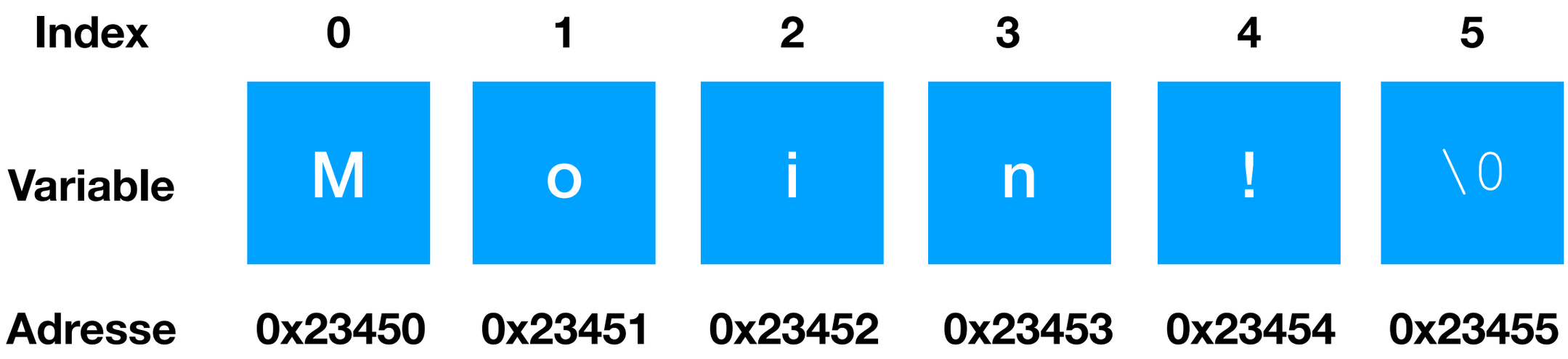
- In C: ein-dimensionale Arrays aus Buchstaben mit einem Null-Byte am Ende.

```c
#include <stdio.h>

int main () {
  const char greeting[]="Moin!";
  printf("Greeting message: %s\n", greeting);
  return 0;
}
```



Zeichenketten in C

```c
#include <stdio.h>

int main () {
   const char moin[]="Moin!";
   const char *greeting=moin;
   printf("Greeting message: %s\n", greeting);
   return 0;
}
```



Zeichenketten in C

```c
#include <stdio.h>

int main () {
   const char moin[]="Moin!";
   const char *greeting=moin;
   printf("Greeting message: %s\n", greeting);
   return 0;
}
```

Zeichenketten in C

```c
#include <stdio.h>
#include <string.h>

int main () {

   char str1[12] = "Hello";
   char str2[12] = "World";
   char str3[12];
   size_t len;

   /* copy str1 into str3. Always use strncopy */
   strncpy(str3, str1, sizeof(str3));
   printf("strncpy(str3, str1, ...):  %s\n", str3);

   len = strlen(str1);
   /* concatenates str1 and str2 */
   strncat(str1, str2, sizeof(str1) - len - 1);
   printf("strncat(str1, str2, ...): %s\n", str1);

   /* total length of str1 after concatenation */
   len = strlen(str1);
   printf("strlen(str1):  %zu\n", len);

   return 0;
}
```

Strings in C++

- In C++: Strings werden auch in Arrays gespeichert
- Aber Speichermanagement passiert automatisch — und dynamisch
- Zuweisungsoperator (=) bewirkt Kopieren der Zeichenkette

```cpp
#include <iostream>
#include <string>

using namespace std;

int main () {
  string str1 = "Hello";
  string str2 = "World";
  string str3;

  // copy str1 into str3
  str3 = str1;
  cout << "str3: " << str3 << endl;

  // concatenates str1 and str2
  str3 = str1 + str2;
  cout << "str1 + str2: " << str3 << endl;

  return 0;
}
```

Strings in C++

```cpp
#include <string>

using namespace std;

int main(){
  // strings erzeugen
  string hallo;
  string moin("Moin");
  string greeting(moin);

  // strings kopieren

  hallo=moin;

  // string manipulieren

  moin.append(greeting);
  moin.insert(4, " ");
  hallo.clear(); // löschen
```

```cpp
  // strings konkatenieren

  hallo = moin + greeting;

  // Zugriff auf Elemente & und Abschnitte

  string m1 = hallo.substr(0,4);
  char m = m1[0];

  // Eigenschaften

  int l = m1.size();
  bool isEmpty = m1.empty();

}
```

Strings in C++

Element access

- `at`
- `operator[]`
- `front` (C++11)
- `back` (C++11)
- `data`
- `c_str`
- `operator basic_string_view` (C++17)

Iterators

- `begin`
  `cbegin` (C++11)
- `end`
  `cend` (C++11)
- `rbegin`
  `crbegin` (C++11)
- `rend`
  `crend` (C++11)

Capacity

- `empty`
- `size`
  `length`
- `max_size`
- `reserve`
- `capacity`
- `shrink_to_fit` (C++11)

Operations

- `clear`
- `insert`
- `erase`
- `push_back`
- `pop_back` (C++11)
- `append`
- `operator+=`
- `compare`
- `starts_with` (C++20)
- `ends_with` (C++20)
- `replace`
- `substr`
- `copy`
- `resize`
- `swap`

Search

- `find`
- `rfind`
- `find_first_of`
- `find_first_not_of`
- `find_last_of`
- `find_last_not_of`

Non-member functions

- `operator+`
- `operator==`
  `operator!=`
  `operator<`
  `operator>`
  `operator<=`
  `operator>=`
- `std::swap`(std::basic_string)
- `erase`(std::basic_string)
  `erase_if`(std::basic_string) (C++20)

Input/output

- `operator<<`
  `operator>>`
- `getline`

https://en.cppreference.com/w/cpp/string/basic_string

Pointer in C

Wozu benötigt man Pointer in C?

Pointer in C

Wozu benötigt man Pointer in C?

```c
int* ptr;
ptr = (int*)malloc(n * sizeof(int));
free(ptr);
```

1. Dynamische Speicherverwaltung

Pointer in C

Wozu benötigt man Pointer in C?

```c
double getAverage(int arr[], int size) {

   int i;
   double avg;
   double sum = 0;

   for (i = 0; i < size; ++i) {
      sum += arr[i];
   }

   avg = sum / size;

   return avg;
}

int main () {

   /* an int array with 5 elements */
   int balance[5] = {1000, 2, 3, 17, 50};
   double avg;

   /* pass pointer to the array as an argument */
   avg = getAverage( balance, 5 ) ;

   /* output the returned value */
   printf( "Average value is: %f ", avg );

   return 0;
}
```

2. Effiziente Parameter-Übergabe (Call-by-Reference)

Pointer in C

Wozu benötigt man Pointer in C?

```c
/* call by value */

int add(int x,y) {
  return (x+y);
}


/* call by reference */

void swap(int *x, int *y) {

   int temp;
   temp = *x;     /* save the value at address x */
   *x = *y;       /* put y into x */
   *y = temp;     /* put temp into y */

   return;
}

int main() {
  int a = 100;
  int b = 200;

  swap(&a,&b);
}
```

3. Übergabe von Ergebnisparametern (“Call by Reference”)

Pointer in C

Wozu benötigt man Pointer in C?

1. Dynamische Speicherverwaltung
2. Effiziente Parameterübergabe (“Call by Reference”)
3. Übergabe von Ergebnisparametern (“Call by Reference”)

Probleme dabei?

Referenzen in C++

1. ~~Dynamische Speicherverwaltung~~
2. Effiziente Parameterübergabe (“Call by Reference”)
3. Übergabe von Ergebnisparametern (“Call by Reference”)

Default (C/C++): Call-by-Value

```
#include <string>
#include <iostream>

using namespace std;

// call by value

void printString(string printString) {
   cout << printString << endl;
}

int main() {
   string moin("Moin");
   printString(moin);
}
```

Wertparameter, `string`-Objekt wird kopiert

Referenzen in C++

```
#include <string>
#include <iostream>

using namespace std;

// call by reference

void printString(const string& printString) {
  cout << printString << endl;
}


int main() {
  string moin("Moin");
  printString(moin);
}
```

Referenz auf `string`-Objekt `moin` wird übergeben

→ Effiziente Parameter-Übergabe

Referenzen in C++

```
#include <string>
#include <iostream>

using namespace std;

// call by reference

void readString(string& eingabe) {
  cin >> eingabe;
}


int main() {
  string mystring;
  readString(mystring);
}
```

Referenz auf `string`-Objekt `mystring` wird übergeben (hier nicht `const`)

→ Übergabe von Ergebnis-Parametern

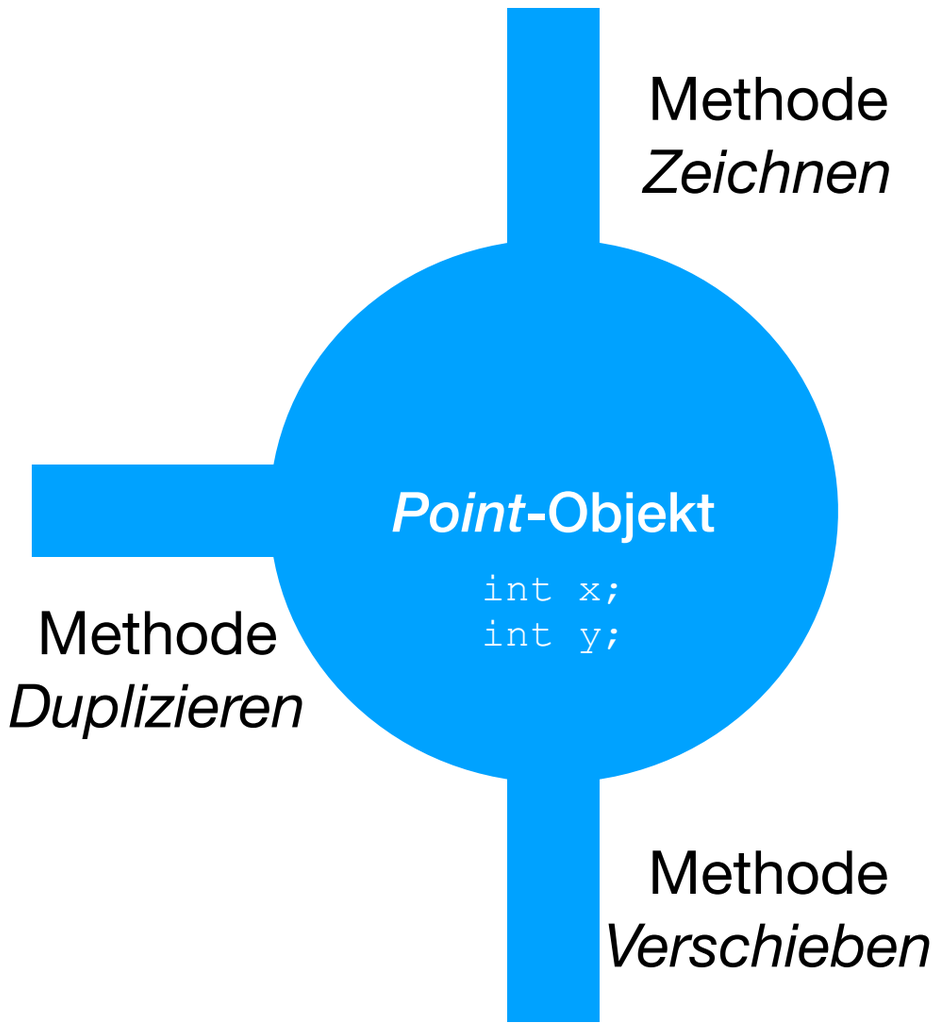
Constness beachten!

```
#include <string>
#include <iostream>

using namespace std;

// call by reference

void readString(string& eingabe) {
  cin >> eingabe;
}


int main() {
  const string mystring;
  readString(mystring);
}
```

Fehler: `mystring` als `const` deklariert

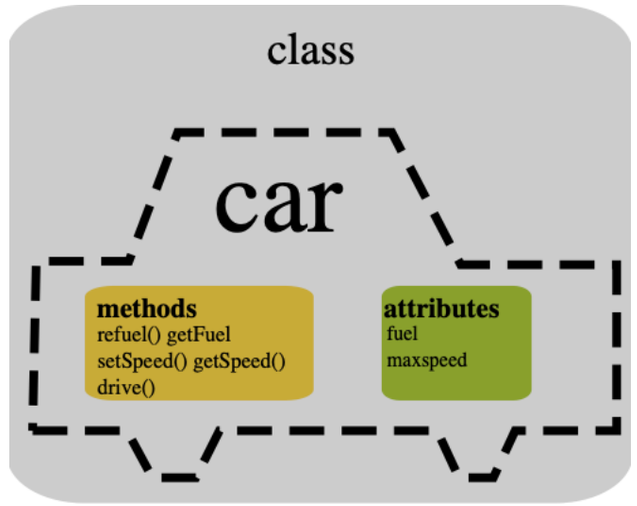
Klassen in C++

- Vollwertige Abstrakte Datentypen
  - Die sich wie “eingebaute” Datentypen verhalten
  - Interne Datenstruktur: ***Kapselung***
  - Extern aufrufbare Funktionen (Methoden)

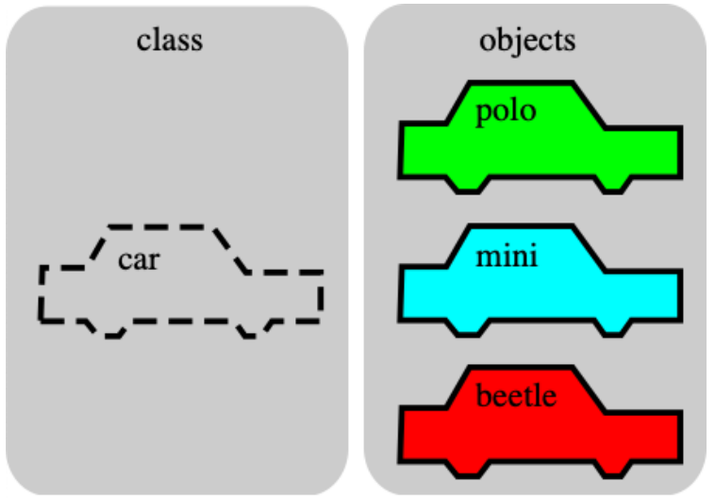


Klassen und Objekte

- Klassen: Definitionen von Datentypen
  - Eigenschaften / Attribute
  - Methoden



- Objekte: Instanzen von Klassen
  - Objekttyp==Klasse
  - Objekte können Attribute ihrer Klassen mit unterschiedlichen Werten versehen



Beispiel

```
class SimpleString {

  char buffer[128];
  int strSize;

};

int main() {
  SimpleString greeting;
  return 0;
}
```

Beispiel

```
#include <iostream>

class SimpleString {

  char buffer[128];
  int strSize;
};

int main() {
  SimpleString greeting;

  std::cout << greeting.strSize;

  return 0;
}
```

Fehler: Feld ist privat

Beispiel

Deklaration der Member-Funktion

Definition der Member-Funktion

Verwendung der Member-Funktion

```
#include <iostream>

class SimpleString {

  char buffer[128];
  int strSize;

public:
  int size();
};

int SimpleString::size() {
  return strSize;
}

int main() {
  SimpleString greeting;

  std::cout << greeting.size();

  return 0;
}
```

Beispiel

```
#include <cstring>
#include <iostream>

class SimpleString {

  char buffer[128];
  size_t strSize;

public:
  void init(const char* initString);
  size_t size(void) const;
};

size_t SimpleString::size(void) const {
  return strSize;
}

void SimpleString::init(const char* initString) {
  strncpy(buffer, initString, sizeof(buffer));
  strSize = strlen(buffer);
}



int main() {
  SimpleString greeting;

  greeting.init("Moin");

  std::cout << greeting.size();

  return 0;
}
```

Konstruktoren

- Klassen benötigten in der Regel eine Form der Initialisierung
  - Initialwert annehmen
  - Member-Objekte initialisieren
  - ggf. Speicher allozieren
- Dafür hat C++ das Sprachelement „Konstruktor“

```
#include <cstring>
#include <iostream>

class SimpleString {

  char buffer[128];
  size_t strSize;

public:
  void init(const char* initString);
  size_t size(void) const;
};

size_t SimpleString::size(void) const {
  return strSize;
}

void SimpleString::init(const char* initString) {
  strncpy(buffer, initString, sizeof(buffer));
  strSize = strlen(buffer);
}



int main() {
  SimpleString greeting;

  greeting.init("Moin");

  std::cout << greeting.size();

  return 0;
}
```

Konstruktoren

Deklaration des Konstruktors

Definition des Konstruktors

Verwendung des Konstruktors

```
#include <string.h>
#include <iostream>

class SimpleString {

  char buffer[128];
  int strSize;

  void init(const char* initString);
public:
  SimpleString(const char* initString);
  size_t size(void) const;
};

size_t SimpleString::size(void) const {
  return strSize;
}

void SimpleString::init(const char* initString) {
  strncpy(buffer, initString, sizeof(buffer));
  strSize = strlen(buffer);
}

SimpleString::SimpleString(const char* initString) {
  init(initString);
}

int main() {
  SimpleString greeting("Moin");

  std::cout << greeting.size();

  return 0;
}
```