# C++ Kurs Teil 3

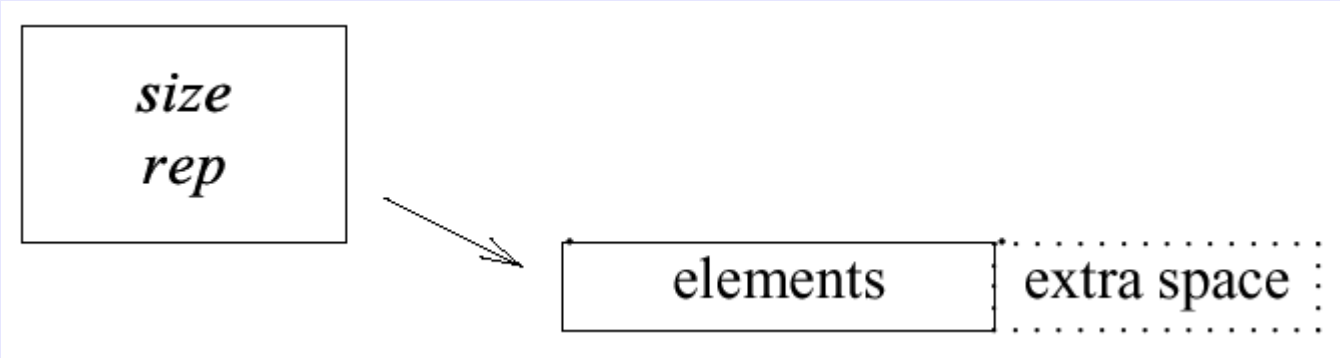
- Standard Template Library (STL)
  - Übersicht
  - vector<>
  - algorithm: sort, for_each
  - map<>
- Kommunikation mit der shell
- Hyper Text Markup Language (HTML)

# Standard Template Library (STL)

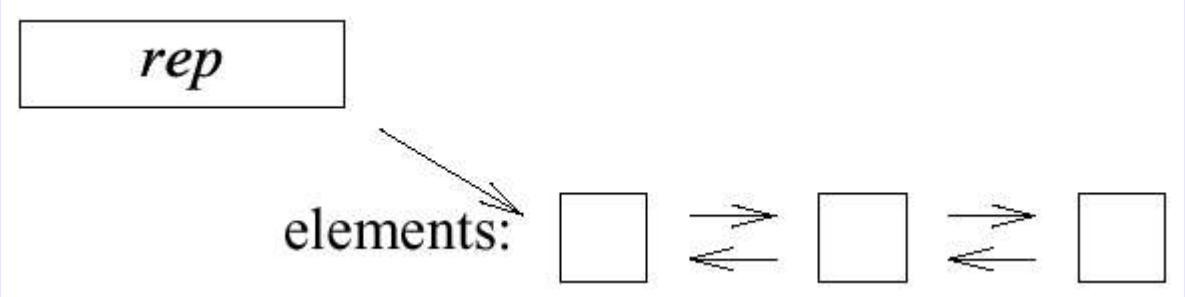
- Containers (vector, list, map, ...)
- General Utilities (date, time, ...)
- Algorithms (sort, for_each, ...)
- Strings (string, wide-character stuff, ...)
- Input/Output (streams, manipulators...)
- Language Support (numeric limits, dynamic memory, ...)
- Numerics (sqrt, sin, cos, complex, ...)

# Übersicht über die Standard Container

- **vector<T>**
  eindimen-
  sionales Array

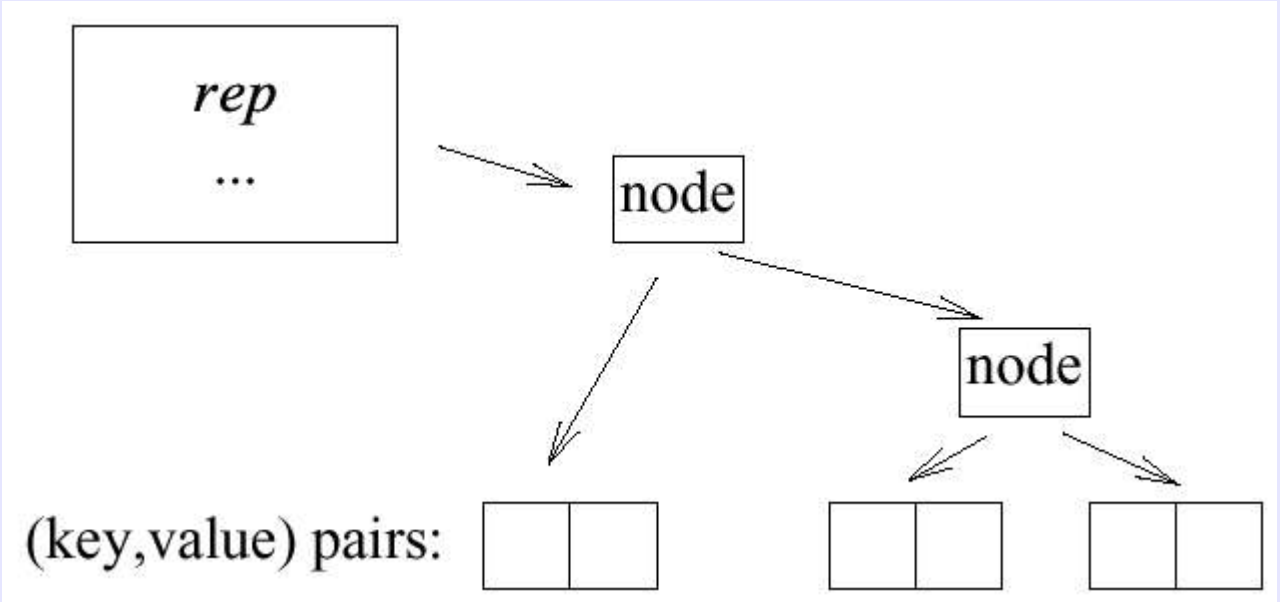



- **list<T>**
  doppelt
  verketteteListe

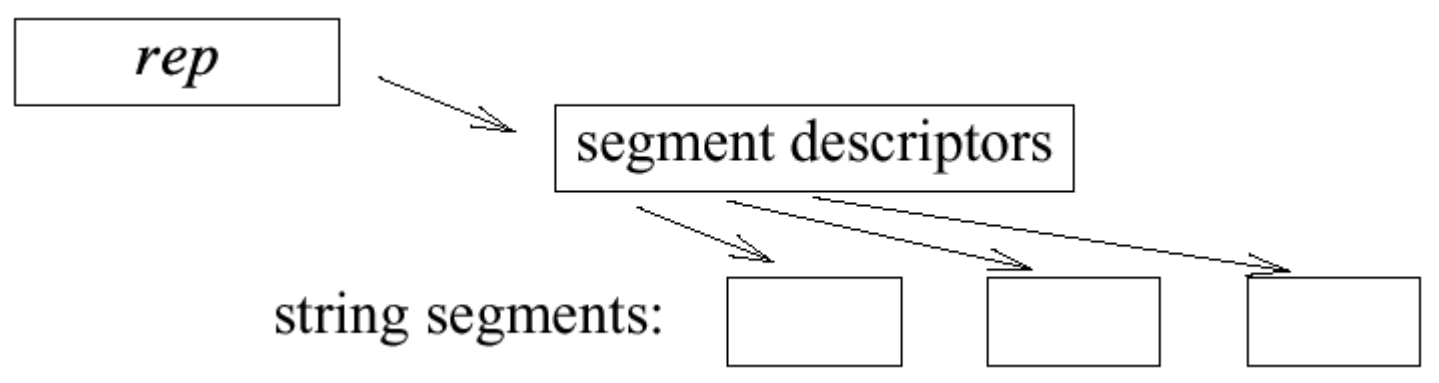


# Übersicht über die Standard Container

- **map<Key,T>** assoziatives Array (Binärbaum)




- **string**: Zeichenkette



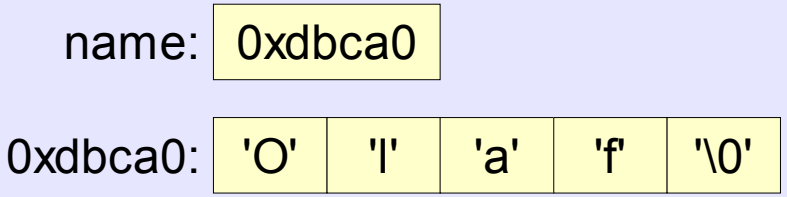
# c-style-strings und std::string

```
/* ein c-style string ist nur ein Pointer! das
Ende des strings ist durch '\0' gekennzeichnet
*/
const char* name = "Olaf";

/* std::string ist eine Klasse. Umwandlung von
c-style in std::string z.B.  durch Zuweisung
*/
std::string name2 = name;

// std::strings können verkettet werden
name2 += ".txt";

/* Umwandlung zurück mit c_str(), z.B. als File
Name für einen ifstream */
std::ifstream file( name.c_str());
```



# Übersicht über die Standard Container

- **stack<T>**
  "last in first out" Container
- **queue<T>**
  "first in first out" Container (fifo)
- **priority_queue<T>**
  teilsortierte Queue (top() liefert immer das größte Element)

- komplette Übersicht in den Referenzen

# STL vector
# (eindimensionales Array)

```
#include <string>
#include <vector>

int main( int argc, char** argv)
{
  std::vector<std::string> people;

  people.push_back( "Chris");
  people.push_back( "Bert");
  people.push_back( "Detlef");
  people.push_back( "Alfons");
  people.push_back( "Gerhard");

  std::cout << "Der 5. heißt: " << people[4] <<std::endl;

  people.resize( 7);
  people[5] = "Fred";
  people[6] = "Ernie";
  return 0;
}
```

Leeres Array aus strings

Einige strings anfügen (Array wächst automatisch)

Zugriff auf das 5. Element

Vergrößern

# Elemente ausgeben und sortieren

```
#include <vector>
#include <string>
#include <algorithm>
....

  // Alle Elemente ausgeben
  std::vector<std::string>::const_iterator p;
  for( p = people.begin(); p != people.end(); ++p)
  {
    std::cout << *p << std::endl;
  }

  // Sortieren
  std::sort( people.begin(), people.end());
```

Iterator (verhält sich wie ein Pointer)

Element, auf das der Iterator zeigt

Quicksort aus 'algorithm'

# Sortieren mit eigener Vergleichs-Funktion

```
// rückwärts sortieren
std::sort( people.begin(), people.end(), myGreaterThan);
```

Die Vergleichsfunktion muß extra definiert werden:

```
bool myGreaterThan( const string &a, const string &b)
{
  return (a > b);
}
```

# Andere Methode zur Ausgabe aller Elemente (for_each)

```
// Alle Elemente ausgeben
std::for_each( people.begin(), people.end(), print_entry);
```

Für jedes Element zwischen begin() und end() die Funktion **`print_entry()`** aufrufen

**`print_entry()`** muß extra definiert werden und bekommt eine Referenz auf das jeweilige Element übergeben

```
void print_entry( const std::string& name)
{
  std::cout << name << std::endl;
}
```

# STL map
# (assoziatives Array)

„Array“ vom Typ int mit string als index (wird intern als balancierter Baum gespeichert)

```
std::map<std::string, int> histogram;

std::ifstream documentFile( "GPL_V2");

std::string word;
while( documentFile >> word)
{
  histogram[word] += 1;
}

for( std::map<std::string, int>::const_iterator
       p = histogram.begin(); p != histogram.end(); ++p)
{
  std::cout << p->first << ": " << p->second << std::endl;
}
```

Datei „GPL_V2“ zum Lesen öfnen

Element-Zugriff: Wenn noch kein Eintrag existiert, wird automatisch ein neuer erzeugt

Ausgabe aller Histogramm-Einträge (automatisch alphabetisch sortiert)

# Übersicht über die Standard Container

| Standard Container Operations | | | | | |
|---|---|---|---|---|---|
| | [] | List Operations | Front Operations | Back (Stack) Operations | Iterators |
| | §16.3.3<br>§17.4.1.3 | §16.3.6<br>§20.3.9 | §17.2.2.2<br>§20.3.9 | §16.3.5<br>§20.3.12 | §19.2.1 |
| *vector* | const | O(n)+ | | const+ | Ran |
| *list* | | const | const | const | Bi |
| *deque* | const | O(n) | const | const | Ran |
| *stack* | | | | const+ | |
| *queue* | | | const | const+ | |
| *priority_queue* | | | O(log(n)) | O(log(n)) | |
| *map* | O(log(n)) | O(log(n))+ | | | Bi |
| *multimap* | | O(log(n))+ | | | Bi |
| *set* | | O(log(n))+ | | | Bi |
| *multiset* | | O(log(n))+ | | | Bi |
| *string* | const | O(n)+ | O(n)+ | const+ | Ran |
| *array* | const | | | | Ran |
| *valarray* | const | | | | Ran |
| *bitset* | const | | | | |

Beispiele: insert() push_front() push_back()

# Kommunikation mit der shell

```
#include <iostream>

int main( int argc, char** argv)
{
  std::cout << "Der Programmname ist: " << argv[0] << std::endl;
  std::cout << "Es wurden " << argc-1 << " Argumente übergeben\n";
  std::cout << "Das 1. Argument ist: " << argv[1] << std::endl;
  std::cout << "Das 2. Argument ist: " << argv[2] << std::endl;
  std::cout << "Das 3. Argument ist: " << argv[3] << std::endl;

  return 0;
}
```

```
> sample_code hallo du da
Der Programmname ist: sample_code
Es wurden 3 Argumente übergeben
Das 1. Argument ist: hallo
Das 2. Argument ist: du
Das 3. Argument ist: da
```

# Doxygen

- automatische Dokumentation
- Klassendiagramme
- erstellt HTML und LATEX Dokumentation
- ...

```
/*! ***************************************************************
**
** \file dummyclass.hh
** \brief provides dummy class for demonstration of
**        doxygen-functionality.
**
*****************************************************************/
#include <parentclass.hh>
  //! brief description of Dummyclass.
  /*! More detailed description can be
      longer. */
class DummyClass : public ParentClass
{
public:
  /*! \brief    a brief description can also last over several lines
   *            as shown in this example of a constructor.
   *
   *  detailed description of the constructor:
   *   creates new Dummyclass Instance.
   *
   *   \param      param1    some initialization parameter
   *   \param      name      name of the instance.
   *
   *   \exception  DummyClassException the possible exceptions
   */
    DummyClass(const int param1, const char *name);
  //!  calculation of some reasonable value.
  /*!
   *   \param      param1    some necessary parameter for calculation
   *
   *   \return     the result of the calculation
   */
    int calc_something(const int param1);
  /*!
      a public variable containing one property.
   */
    int property1;
    float property2; //!< this is a backward-reference documentation line.
};
```