Die Standard Template Library
STL

Erste Einblicke

STL

- Motivation
- Grundbegriffe
- Beispiel Vektoren

Motivation:

- Die Standard-Template-Library ist erst in den letzten Jahren in den C++ Standard aufgenommen worden.
- "Die STL ist ein komplexes Stück Software-Engineering, das die raffiniertesten Features von C++ verwendet."
- Achtung: die STL baut auf Zeiger, Referenzen und Templates auf.

- Ziel:
  - Überblick und einige Beispiele

Grundbegriffe

- Container
  sind Objekte, die andere Objekte enthalten. Beispiele sind:
  - vector als dynamisches Array
  - deque (Warteschlange) , List (lineare Liste)
- Algorithmen
  arbeiten mit den Containern. Beispiele:
  - sortieren, suchen,…
- Iteratoren
  sind Objekte. Sie geben (wie Zeiger) die Möglichkeit in Containern zu navigieren. Beispiele:
  - Forward, Input, Random Access,…

Vector

- Aufruf als Template
  - vector<typ>
- Klasse stellt nützliche Funktionen bereit wie:
  - size() - aktuelle Grösse
  - push_back() – am Ende anfügen
- Klasse erweitert dynamisch falls der anfängliche Speicherplatz (v(10)) ausgeht

```
#include <vector>

int main()
{
  vector<char> v(10);
  int i;

  cout << "Laenge:" << v.size();

  for (i=0;i<10;i++) v[i] = i +'a';

  for (i=0;i<10;i++) cout << v[i];

  // dynamisches Erweitern
  for(i=0;i<10;i++)
    v.push_back(i+10+'a');

  cout << "Laenge " << v.size();
```

Zugriff über Iteratoren

- Erzeugen des Iterators
  - vector<typ>::iterator p;
- Der Iterator wird verwendet wie ein Zeiger. D.h.
  - p++ , p– verändert Position
  - *p spricht den Inhalt an der Position an .
    (entsprechend des Datentyps)
- Mit zusätzlichen Funktionen kann der Zeiger initialisiert (beeinflusst) werden
  - .begin(), .end()
- Weitere Funktionen
  - insert() (Einfügen an Stelle)
  - erase() (Löschen eines Bereichs)
  - clear() (Löschen aller Elemente)
  - front(), back() (Ref. auf erstes/letztes Element)

```
#include <vector>

int main()
{
  vector<char> v(10);
  vector<char>::iterator p;
  int i=0;

  p = v.begin();
  while (p != v.end()) {
        *p = i + 'a';
        p++;
        i++;
  }
```

Liste

- Aufruf als Template
  - list<typ>
- Klasse stellt nützliche Funktionen bereit wie:
  - push_back() – am Ende anfügen
- Erzeugen des Iterators
  - vector<typ>::iterator p;
- Der Iterator wird verwendet wie ein Zeiger. D.h.
  - p++ , p– verändert Position
  - *p spricht den Inhalt an der Position an .
    (entsprechend des Datentyps)
- Mit zusätzlichen Funktionen kann der Zeiger initialisiert (beeinflusst) werden
  - .begin(), .end()

```
#include <list>

int main()
{
  list<int> l1,l2;
  int i;

  for (i=0; i<10;i+=2)
      l1.push_back(i);
  for (i=1; i<11;i+=2)
      l2.push_back(i);

  list<int>::iterator p =
  l1.begin();
  while( p != l1.end()) {
    cout << *p << " " ; p++; }
}
```

Zusatzfunktionen

- Weitere Funktionen
  - merge() (Mischen zweier Listen)
  - sort() (sortieren)
  - clear() (Löschen aller Elemente)
  - remove() (Einen Wert aus Liste löschen)
  - insert(),…
  - empty() (ist Liste leer?)

```
#include <list>

int main()
{
    list<int> l1,l2;
    int i;

    for (i=0; i<10;i+=2)
        l1.push_back(i);
    for (i=1; i<11;i+=2)
        l2.push_back(i);

    l1.merge(l2);

    list<int>::iterator p =
    l1.begin();
   while( p != l1.end()) {
     cout << *p << " " ; p++; }

}
```