Templates

- Templates werden in deutschen Fachbüchern mitunter als Schablonen bezeichnet.
- Templates dienen der Generierung von Klassen, insbesondere von Containerklassen oder Collections.
- Typische Vertreter sind Vektoren/Arrays, Hashtables, Listen.
- Templates kann man sich wie intelligente Macros, die die Sichtbarkeits- und Typregeln von C++ berücksichtigen, vorstellen.
- Ein Template ist gewissermaßen eine Konstruktions-/Generierungsvorschrift für Klassen.
- Der zu verwendende Typ wird als Klassenparameter angegeben.

- Jedem Template wird `template <class T>` vorangestellt.
- Innerhalb des Templates wird T als bekannter Datentyp verstanden.
- Der Datentyp der generierten Klasse besteht aus TemplateName<Typ>
- Der Name des Konstruktors im Template besteht nur aus dem TemplateNamen.
- Werden MemberFunktionen außerhalb der „Klassenbeschreibung“ des Templates vereinbart, ist zu beachten:
  - Die Herstellung des Klassenbezugs erfolgt über `TemplateName<Typ>::` für jede Funktion
  - **Alle Funktionen, auch außerhalb der Templatevereinbarung sind im HeaderFile anzuordnen.**

Template

```
template <class T>

class Array
{
   public:
       class Range{};
      Array (int Gr = SizeArr);
      Array (const Array&);
      ~Array() {delete theData;}
      Array & operator = (const Array&);
      T& operator [] (int i);
      int  GetNum()const {return Size;}
      void Resize(int NewSz);
   protected:
      int  Size;
      T*   theData;
};
```

Funktionsimplementationen im .h – File

Konventionelle Klasse

```
const int SizeArr = 24;

class IntArr
{
   public:
      class Range{};
      IntArr (int Gr = SizeArr);
      IntArr (const IntArr&);
      ~IntArr() {delete IA;}
      IntArr & operator = (const IntArr&);
      int& operator [] (int i);
      int GetNum()const {return Size;}
      void Resize(int NewSz);
   protected:
      int  Size;
      int *IA;
};
```

Funktionsimplementationen im .cpp –File

```
template <class T>
Array<T> :: Array (int Sz)
{
  Size = Sz;
  theData = new T [Size];
  for (int i=0; i<Sz; ++i) theData[i]=0;
}
```

```
IntArr :: IntArr (int Sz)
{
  Size = Sz;
  IA = new int [Size];
  for (int i=0; i<Sz; ++i) IA[i]=0;
}
```

```
template <class T>
Array<T> :: Array (const Array<T>& Other)
{
   Size = Other.Size;
   theData = new T[Size];
   for (int i=0; i<Size;++i)
           theData[i]=Other.theData[i];
}
```

```
IntArr :: IntArr (const IntArr& Other)
{
   Size = Other.Size;
   IA = new int[Size];
   for (int i=0; i<Size;++i)
          IA[i]=Other.IA[i];
}
```

```
template <class T>
Array<T>& Array<T> :: operator = (const
Array<T>& Other)
{
  delete theData;
  theData = new T [Other.Size];
  Size = Other.Size;
  for ( int i=0; i<Size;i++)
          theData[i]=Other.theData[i];
  return * this;
}
```

```
IntArr& IntArr :: operator = (const IntArr&
Other)
{
  delete IA;
  IA = new int [Other.Size];
  Size = Other.Size;
  for ( int i=0; i<Size;i++)
        IA[i]=Other.IA[i];
  return * this;
}
```

```
template <class T>
T& Array<T> :: operator [](  int i)
{
   if (i<Size && i>=0) return
theData[i];
   else
   {
       throw Range();
   }
}
```

```
int& IntArr :: operator [](  int i)
{
   if (i<Size && Size>=0) return IA[i];
   else
   {
       throw Range();
   }
}
```

```
int main()
{
  set_new_handler(MemError);
  {
    Array<int> IA(10);
    try
    {
      int i;
      for (i = 0; i < IA.GetNum() ; ++i)
           IA[i]=i;
      for (i = 0; i < IA.GetNum() ; ++i)
           cout << IA[i] <<", ";
      cout << '\n';
      IA.Resize(16);
      for (i = 0; i < IA.GetNum() ; ++i)

           cout << IA[i] <<", ";
      cout << '\n';
      cout << IA[100] <<'\n';
    }
    catch(Array<int>::Range)
    {
      cerr<< "Indexerror detected\n";
      //exit (1);
    }
  }
```

```
int main()
{
  set_new_handler(MemError);
  {
    IntArr IA(10);
    try
    {
      int i;
      for (i = 0; i < IA.GetNum() ; ++i)
           IA[i]=i;
      for (i = 0; i < IA.GetNum() ; ++i)
           cout << IA[i] <<", ";
      cout << '\n';
      IA.Resize(16);
      for (i = 0; i < IA.GetNum() ; ++i)
           cout << IA[i] <<", ";
      cout << '\n';
      cout << IA[100] <<'\n';
    }
    catch(Array<int>::Range)
    {
      cerr<< "Indexerror detected\n";
      //exit (1);
    }
  }
```

Templatefunktion

```
template <class T>
void sort (Array<T>& A, int n)
{
  //int n=A.GetNum();
  for (int i=0;i<n-1;i++)
    for (int j=i+1; j<n; j++)
      if (A[i]>A[j])
      {
         T temp=A[i];
         A[i]=A[j];
         A[j]=temp;
      }
}
```

Exceptions

- Exceptionhandling dient der Behandlung von Ausnahme- und Fehlersituationen.
- Dabei werden Bereiche, in denen Exceptions möglicherweise auftreten, in try/catch -blöcke gekapstelt.
- Mit Hilfe der catch-klauseln werden Exceptions aufgefangen und ggf. behandelt.
- Tritt eine Exception auf, so wird der try-block verlassen und zu einer passende catch-klausel verzweigt, dabei wird der Stack ggf. bereinigt.

Exceptions

- Die zu werfenden Exceptions können in C++ von beliebigem Datentyp sein.
- Im Beispiel 1 wird eine Klasse Range als Exception gesendet, im Beispiel 2 ein Objekt der Klasse Range mit dem Wert des fehlerhaften Index.
- Mit **`throw`** werden Exceptions bei Bedarf „geworfen“, um eine Ausnahme zu signalisieren.

Exceptions

```
template <class T>
class Array
{
   public:
     class Range
     {
         int i;
         public:
             Range(int i){this->i=i;}
             int getI(){return i;}
    };
     . . .
```

Exceptions

```
template <class T>
T& Array<T> :: operator [](  int i)
{
   if (i<Size && i>=0) return theData[i];
   else
   {
       throw new Range(i);
   }
}
```

Exceptions

Allgemeiner Aufbau:

```
try
{

…… Text, in dem Exceptions auftreten können

}
catch(ex1){...Code zur Behandlung...}
catch(ex2){...Code zur Behandlung…}

. . .
```

Exceptions

```
#include <iostream>
#include <typeinfo>

using namespace std;
class myECl{};

class myExO
{
  public:  myExO(int x){reason=x;}
           int what(){return reason;}
  private: int reason;
};

class myExD:public exception{};
```

Exceptions

```
int main(int argc, char** argv)
{
  try
  {
    switch (argv[1][0])
    {
      case 'i':throw 77;                        break;
      case 'c':throw 'X';                       break;
      case 'p':throw "Lieblingsexception"; break;
      case 't':throw myECl();                   break;
      case 'o':throw myExO(789);                break;
      case 'd':throw myExD();                   break;
    }
  }
```

Exceptions

```
  catch (int   i)
   {cout <<"Exception int   :"<<i<<endl;}
  catch (char  c)
   {cout <<"Exception char  :"<<c<<endl;}
  catch (const char* p)
   {cout <<"Exception char* :"<<p<<endl;}
  catch (myECl t)
   {cout <<"Exception Object :"
         <<typeid(t).name()<<endl;}
  catch (myExO&o)
   {cout <<"Exception Object:"<<o.what()<<endl;}
  catch (myExD&d)
   {cout <<"derived Exception:"<<d.what()<<endl;}
}
```