Die Klasse „string“

Verfasser: Christian Bartl

Index

1. Allgemein

Strings kann man in C++ entweder wie in C in char-Arrays oder in Zeigern auf char ablegen oder in Objekte der in C++ verfügbaren Klasse „string“. Diese erlaubt auf einfache Weise beliebig lange und dynamische Strings zu verwalten. Zudem sind viele Operationen deutlich einfacher und gefahrloser anzuwenden als in C. Zur Benutzung der Klasse string muss man die Headerdatei „string“ einbinden über den Befehl: #include <string>. Achtung nicht verwechseln mit cstring und string.h.
Ein Objekt der Klasse string wird durch Konstruktoren erzeugt.

- Default Konstruktor (leerer String)
  string Name;

- Typumwandlungskonstruktor von „const char *“ in „string“
  string Name („Hallo“);

- Typumwandlungskonstruktor
  String Name = „OK“;

- Kopierkonstruktor
  String Name (Name1);

- Kopierkonstruktor
  String Name = Name1;

- String enthält als X. Zeichen den Buchstaben ’u’
  String Name (X, ’u’);

- char-Array wird als „char *“ interpretiert und in dem string-Objekt Name abgelegt
  char Name [6] = “Hallo“;
  string str_Name (Name);

- char-Zeiger wird in string-Objekt Name abgelegt
  char * Name = “Wort“;
  string str_Name (Name);

Wichtig ist, dass zuerst immer ein leerer String erzeugt wird der dann automatisch vergrößert bzw. wieder verkleinert wird. Eine Längenangabe sowie ein manuelles Vergrößern des Strings ist nicht notwendig.

2. Eingabe von Strings

Strings kann man entweder mit „cin>>“ eingeben oder mit der globalen Funktion „getline(…)“.

Da „cin>>“ führende Leerzeichen überliest und nach Beginn des Einlesen des 1.Zeichens stoppt, wenn es auf folgende Leerzeichen stößt, ist es gut geeignet um Strings getrennt einzugeben. Gibt man allerdings ein Wort – Leerzeichen – Wort2 ein,

wird nur Wort gespeichert. Nicht zu vergessen ist das „cin>>“ das abschließende Newline (also Enter) am Ende der Zeile im Tastaturpuffer lässt.

```
string Name;
cin>> Name;
```

Der Befehl „getline kombiniert mit „cin“ überliest führende Leerzeichen nicht und stoppt die Eingabe, wenn im Eingabepuffer Newline steht (also die Enter-Taste gedrückt wurde). Daher ist der Befehl gut geeignet um Daten die durch Leerzeichen getrennt sind, in einen String einzulesen.

```
string Name;
getline (cin, Name);
```

Um 2 oder mehr Werte mit einer Eingabe abzufragen aber in verschiedene Strings zu schreiben gibt es die Möglichkeit „getline(…)“ ein Argument mitzugeben. Zum Beispiel sollen Vor- und Nachname die durch ein Komma getrennt werden über eine Abfrage in den String Vorname und Nachname geschrieben werden.

```
string Vorname, Nachname;
getline (cin, Vorname, ’,’);
getline (cin, Nachname, ’,’);
```

Dies bedeutet nichts anderes, als das bis zum Komma eingelesen wird, dass Komma aber nicht in einen der beiden Strings geschrieben wird und dann wieder weiter eingelesen wird, bis ein Newline folgt.

3. Ausgabe von Strings

Die Ausgabe erfolgt über den Befehl „cout<<“.

```
String Name;
cin>> Name;
cout<< Name;
```

4. Länge eines Strings

Die Länge eines Strings kann durch die Methode „size ()“ ermittelt werden. Sie liefert die Länge als Wert des Datentyps „string :: size_type“ zurück (Dies ist im Prinzip ein vorzeichenloser ganzzahliger Wert).

```
string :: size_type Laenge;
string Name;

Laenge = Name.size();                    //Laenge = 0

Name =  “Hallo“;
Laenge = Name.size();                    //Laenge =  5
```

5. Durchlaufen aller Zeichen eines Strings

Die Zeichen eines string-Objekts sind Variablen vom Datentyp char. Die einzelnen Zeichen können mit einer for-Schleife durchlaufen werden. Da ein string-Objekt seine Länge gespeichert, hat benötigt man keine Variable, die die Länge speichert. Um auf

die Zeichen zugreifen zu können, verwendet man wie bei Arrays eckige Klammern „[]“ (ohne Indexcheck) oder die Methode „at(…)“ (mit Indexcheck).

Beispiel ohne Indexcheck

```
string Name = “Wert“;
string :: size_type i;

for (i=0; i<Name.size(); i++)
{
        cout<< Name [i];                    //Ausgabe
        Name [i] = ’0’                      //Eingabe
}
```

Beispiel mit Indexcheck

```
string Name = “Wert“;
string :: size_type i;

for (i=0; i<Name.size(); i++)
{
        cout<< Name.at (i);                 //Ausgabe
        Name.at (i) = ’0’                   //Eingabe
}
```

Bei Verwendung der Methode „at(…)“ können im Gegensatz zur Möglichkeit wie bei Arrays Zugriffsfehler in Form des Abbrechens des Programms erkannt werden.

6. Kopieren von Strings

Das Kopieren der string-Objekte mit der Wertzuweisung erzeugt eigenständige Kopien der Zeichenkette. Dies gilt auch bei Wertzuweisung con char-Arrays oder char-Zeigern.

```
string Name1= “Hallo“, Name2;
Name2 = Name1;
        // Name 2 ist ein von Name 1 unabhängiges Objekt, das aber dieselbe
           Zeichenkette verwaltet

char * Name1 = “Wort“;
string Name2;
Name2 = Name1;
        // Name2 ist ein string-Objekt, das die Zeichenkette “Wort“ als String
           verwaltet.

string Name;
Name= ’a’;
        // Name ist ein string-Objekt, das nur das Zeichen ’a’ als String
           verwaltet.
```

```
String Name = ’a’;
        // Achtung FALSCH ! Die Initialisierung eines string-Objekts mit
        einem char-Wert ist nicht erlaubt.
```

Sollen nicht alle sondern nur die Zeichen von x bis y kopiert werden, erfolgt dies durch die Methode „substr(…)“. Dabei wird als erster Parameter die Nummer des Anfangszeichens und als zweiter Parameter die Nummer des Endzeichens angegeben. Werden dabei zu große Werte angegeben, wird maximal bis zum letzten Zeichen kopiert. Die Klasse string liefert eine vorgefertigte Konstante „string :: npos“ die die Anzahl der Zeichen eines string-Objektes enthält.

```
string Name = “Wert Wert Wert“, Name2;

Name2 = Name1.substr(0, 3);
        // Kopiert die Zeichen von 0 bis 3

Name2 = Name1.substr(5, 7);
        // Kopiert die Zeichen von 5 bis 7

Name2 = Name1.substr(4,  string::npos);
        // Kopiert die Zeichen von 4 bis zum Ende des Strings

Name2 = Name1.substr(0,  0);
        // Kopiert ab dem 1.Zeichen nur ein Zeichen des Strings also nur “ “

Name2 = Name1.substr(0,  string::npos);
        // Kopiert alle Zeichen des Strings
```

Um Zeichen in einem String zu ersetzen gibt es die Methode “replace(…)”. Die Übergabeparameter sind wieder Anfangsnummer und Endnummer. Hinzu kommt jetzt der Wert oder der String, der eingesetzt werden soll.

```
string Name = “Wert Wert Wert“, Name2 = “Wert1“;

Name.replace(0, 3, “Hal“);
        // Ersetzt die Zeichen von 0 bis 3 durch “Hal“

Name.replace(0, 4, Name2);
        // Ersetzt die Zeichen von 0 bis 4 durch den längeren String Name2. Es
           steht nun in Name “Wert1 Wert Wert“
```

7. Verketten von Strings

Man kann string-Objekte auch verketten also zusammenhängen mit dem überladenen „+“ – Operator. Dabei ist es auch erlaubt char-Arrays, char-Zeiger sowie char-Werte zur Verkettung zu benutzen.

```
string Name1 = “Wert1“;
string Name2 = “Wert2“;
string Name3;
```

```
Name3 = Name1 + Name2;

Char * Wort1 = “Wort1“;
Name3 = Name1 + Wort;

Char Wort2 [5] = “Wort2“;
Name3 = Name1 + Wort2;

Name3 = Name1 + “Wort3“;

Name3 = Name1 + ’W’;
```

Um nur eine bestimmte Anzahl von Zeichen eines string-Objekts anzuhängen gibt es die Methode „append(…)“. Als erstes Argument übergibt man den anzuhängenden Wert oder String. Als zweites Argument wird die Anzahl der zu kopierenden Zeichen x übergeben, das heißt es werden die Zeichen 0 bis x angehängt. Übergibt man dieses Argument nicht werden alle Zeichen kopiert.

```
string Name1 = “Wert1“;
string Name2 = “Wert2“;;

Name1.append(“, “);
        //Es wird “, “ angehängt

Name1.append(Name2);
        //Es wird “Wert2“ angehängt

Name1.append(Name2, 3);
        //Es werden nur 3 Zeichen angehängt also “Wer“
```

8. Vergleichen von Strings

String-Objekte kann man einfach mit dem „==“ Operator vergleichen. Es ist auch möglich char-Arrays, char-Zeiger aber nicht char-Werte beim Vergleich zu benutzen.

```
String Name1 = “Wert“, Name2;
Name2 = Name1;

if (Name1 == Name2)
        cout<<“Strings sind gleich“;
                // Vergleichsergebnis ist True

bool Vergleich;
vergleich = Name1 == Name2;
                // Die Bool’sche Variable enthält in diesem Fall den Wert True;

char * Wort = “Wert“;
if (Name1 == Wort)
        cout<<“Strings sind gleich“;
                // Vergleichsergebnis ist True
```

```
char Wort[5] = “Wert1“;
if (Name1 == Wort)
        cout<<“Strings sind gleich“;
                // Vergleichsergebnis False

Name1 = ’a’;
if ( Name1 == ’a’)
        cout<<“Strings sind gleich“;
                // Achtung FALSCH ! Vergleich von einzelen char-Werten ist
                   nicht erlaubt
```

Es ist nicht nur möglich auf Gleichheit zu prüfen, sondern auch auf kleiner, kleiner gleich, größer und größer gleich. Dabei werden die Inhalte der beiden zu vergleichenden string-Objekte lexikographisch (bezüglich der Zeichensatznummern) verglichen.

Operationen für die zwei string-Objekte s1 und s2

```
s1 > s2         // true, falls s1 lexikographisch größer als s2 ist
s1 > s2         // false, falls s1 nicht lexikographisch größer als s2 ist

s1 == s2        // true, falls s1 lexikographisch mit s2 übereinstimmt
s1 == s2        // false, falls s1 nicht lexikographisch mit s2 übereinstimmt

s1 < s2         // true, falls s1 lexikographisch kleiner als s2 ist
s1 < s2         // false, falls s1 nicht lexikographisch kleiner als s2 ist

s1 >= s2        // true, falls s1 lexikographisch größer als s2 ist oder mit s2
                   übereinstimmt
s1 >= s2        // false, falls s1 lexikographisch kleiner als s2 ist

s1 <= s2        // true, falls s1 lexikographisch kleiner als s2 ist oder mit s2
                   übereinstimmt
s1 <= s2        // false, falls s1 lexikographisch größer als s2 ist

s1 > s2         // true, falls s1 lexikographisch größer als s2 ist
s1 > s2         // false, falls s1 nicht lexikographisch größer als s2 ist
```

Das Vergleichen zweier string-Objekte ist auch über die Methode „compare(…)“ möglich. Als erstes Argument wird der zweite String übergeben. Als zweites und drittes Argument können optional die Nummer des Anfangszeichens und Endzeichens, die zu vergleichen sind, angegeben werden. Das Vergleichen von verschieden langen Strings ist möglich. Außerdem wird zwischen Groß- und Kleinschreibung unterschieden. Dabei gibt die Funktion folgende Werte zurück:

```
>0          //falls s1 lexikographisch größer als s2 ist
== 0        //falls s1 mit s2 übereinstimmt
<0          //falls s1 lexikographisch kleiner als s2 ist
```

```
string s1 = “Wert1“;
string s2 = “wert1“;

if  (s1.compare(s2) < 0)
        cout<<“s1 ist lexikographisch kleiner als s2“;
            // Vergleichsergebnis True

if  (s1.compare(s2, 1, 4) == 0)
        cout<<“s1 stimmt in den 1 bis 4 Zeichen mit s2 überein“;
            // Vergleichsergebnis True
```

9. Zeichen suchen in Strings

Um in einem String oder einem Teil eines Strings zu suchen verwendet man die Methode „find(…)“. Als erstes Argument wird der zu suchende Wert mit übergeben. Als zweites Argument wird die Position an der zu suchen begonnen werden soll mit übergeben. Wird dieses Argument nicht übergeben, wird automatisch bei Position 0 zu suchen begonnen. Außerdem benötigt man eine Variable vom Datentyp „string :: size_type“ die die Position des Zeichens speichert. Bei nicht vorhanden sein des Zeichens liefert die Funktion den Wert „size :: npos“ (no position) zurück.

Suche nach einem Zeichen von links beginnend

```
string Name = „Wert“;
string :: size_type Position;

Position = Name.find(’e’);
        // Suche nach dem Zeichen ’e’ ab Position 0

Position = Name.find(’e’, 3);
        // Suche nach dem Zeichen ’e’ ab Position 3
```

Suche nach einem Zeichen von rechts beginnend

```
string Name = „Wert“;
string :: size_type Position;

Position = Name.find(’e’);
        // Suche nach dem Zeichen ’e’ ab Position Länge des Strings-1

Position = Name.find(’e’, 3);
        // Suche nach dem Zeichen ’e’ von rechts her ab Position 3
```

Um nach einem von mehreren möglichen Zeichen, von links her beginnend, in einem String zu suchen gibt es die Methode „find_first_of(…)“. Dabei wird als erstes Argument die zu suchenden Zeichen angegeben oder natürlich auch ein String. Als zweites Argument kann wieder die Position, an der zu suchen begonnen werden soll, angegeben werden. Bei nicht vorhanden sein des Zeichens liefert die Funktion den Wert „size :: npos“ (no position) zurück.

```
string Name = “Wert 1234“;
string Zahlen = “0123456789“;
string :: size_type Position;

Position = Name.find_first_of(Zahlen);
        // Position enthält die Position jenes Zeichens das als erstes mit einem
           der Zeichen vom String Zahlen übereinstimmt

Position = Name.find_first_of(„3456“, 2);
        // Es wird erst bei Position 2 zu suchen begonnen
```

Um nach einem Teilstring, von links her beginnend, zu suchen wird wieder die Methode „find(…)“ benutzt. Um nach einem Teilstring, von rechts her beginnend, zu suchen wird wieder die Methode „rfind(…)“ benutzt. Als Argument wird dabei ein ganzer String übergeben. Als zweites Argument kann wieder die Startposition angegeben werden. Bei nicht vorhanden sein des Strings liefert die Funktion den Wert „size :: npos“ (no position) zurück.

```
string Name = “Wert besteht aus einem Wort“;
string Wort = “aus“;
string :: size_type Position;

Position = s.find(Wort);
        // Position enthält die Anfangsposition des Strings nach dem gesucht
           wurde

Position = s.find (“besteht“, 3);
        // Es wird erst bei Position 3 zu suchen begonnen
```

10. Strings umwandeln in andere Datentypen

Man kann Strings durch verschiedene Methoden in char-Arrays oder C-Strings umwandeln.

Um eines Strings in einen C-String umzuwandeln, benötigt man einen C- String vom Typ „const char *“ und die Methode „c_str(void)“. Außerdem wird am Schluss die Binäre Null angehängt.

```
const char * Name;
string Wort = “Wert“;

Name = Wort.c_str();
```

Um einen Strings in ein char-Array umzuwandeln, benötigt man ein Feld vom Typ „const char *“ und die Methode „data(void)“. Es wird aber keine Binäre Null in das Array geschrieben.

```
const char * Name [5];
string Wort = “Wert“;

Name [5] = Wort.data();
```