C-String to cout

```
CString cs("Hello");
wcout << (const wchar_t*) cs << endl;
```

---

```
using namespace std;
#include <iostream>
#include <iomanip>

int main(){
  cout<<setfill('-') <<setw(15)<<endl;

  cout<<dec<<20<<endl;
  cout<<hex<<20<<endl;
  cout<<oct<<20<<endl;
}
```

**Status/Verhalten von Objekten, Member** Objekte, die dynamisch mit new erzeugt worden sind, müssen mit delete vernichtet werden. **Membersichtbarkeit** private ist default. protected nur bei Vererbung. **Erzeugung/Vernichtung von Objekten (Konstrktoren/Destruktoren)** Kopierkonstruktor: ein spezieller Konstruktor, der eine Referenz auf ein Objekt desselben Typs als Parameter entgegennimmt und die Aufgabe hat, eine Kopie des Objektes zu erstellen.
Basisinitialisierer:
Bei der Basisinitialisierung kann ein beliebiger Konstruktor (public oder protected) der Basisklasse benutzt werden. Wird kein Basisinitialisierer angegeben, so wird der Default-Konstruktor der Basisklasse für die Konstruktion verwendet (der dann existieren muss!) **statische Member** STATIC MEMBERDATEN: Für statische Memberdaten existiert nur eine Instanz für alle Objekte der Klasse. Die statischen Memberdaten werden in der Klasse nur deklariert und außerhalb der Klassendeklaration mit vorangestelltem Klassennamen und scope-operator definiert, und müssen auch in ihrer Definition initialisiert werden. Sie werden nicht über die Konstruktoren initialisiert. Auf statische Memberdaten kann zusätzlich über den Klassennamen und den scope-operator zugegriffen werden.
FRIEND-FUNCTIONS: Friend-Funktionen sind keine Memberfunktionen, können aber trotzdem auf die privaten Daten der Klasse, die sie als Friend ausweist zugreifen, sie werden vor allem zur Konvertierung und zum Operatorüberladen benutzt. Um eine Friendfunktion zu vereinbaren, wird die Funktionsdeklaration in die Klassenvereinbarung aufgenommen und das Schlüsselwort friend vorangestellt.
FRIEND-CLASSES: Memberfunktionen von Friend-Klassen haben uneingeschränkten Zugriff zu den privaten Member der Klasse, die die Friend-Vereinbarung enthält. Eine Friend-Klassenvereinbarung besteht aus dem Schlüsselwort friend class gefolgt von dem Klassenamen der befreundeten Klasse. Klassen können auch wechselseitig befreundet sein. Auch einzelne Funktionen können als friend gekennzeichnet werden. **Basisklasse/abgeleitete Klasse**

```
//  Basisklasse
class Person{
  string name;
  //...
};

//  Unterklasse
class Mitarbeiter : Person{
  long sozialversicherungsNr;
  //...
};
```

**Mehrfachvererbung**

```
class A{
  int x;
  //...
};

class B {
  double y;
  //...
};

class C : public A, public B{
  char z;
  //...
};
```

**Überladung unärer/binärer Operatoren** kann nicht überladen werden: . .* :: ?: sizeof
Operatorprioritäten bleiben erhalten

```
class K1{
  public:
    K1() {}
    K1(float real, float imag)     {
      m_real = real;
      m_imag = imag;
    }
    ~K1() {}

    K1 operator+(const K1 &o) const{
      return (K1(o.m_real + m_real, o.m_imag + m_imag));
    }

  private:
    float m_real, m_imag;
};
```

**Templates**

```
template <typename T>
T max(T x, T y){
  if (x < y)
    return y;
  else
    return x;
}
```

Klassentemplate:

```
template <class T>

class Liste  {
  public:
    Liste(); // Eine neue leere
                Liste generieren
    ~Liste() { delete root; } // Liste
                löschen
    void einfuegen(T k);
    // Objekt vorne einfügen
    private:
      Listenelement<T>* root;
      // Wurzel der Liste geht
         niemand was an
};
```

**Dateiarbeit** Schreiben:

```
#include <fstream>
using namespace std;

int main(){
  fstream f;
  f.open("test.dat",
          ios::out);
  f << "Dieser Text geht
       in die Datei"
       << endl;
  f.close();
}
// ios::out schreiben
// ios::in  lesen
// ios::out
// ios::app anhängen
```

Lesen:

```
#include <fstream>
#include <iostream>
using namespace std;

int main(int argc, char *argv[]){
  fstream f;
  char cstring[256];
  f.open(argv[1],
    ios::in);
  while (!f.eof())
  {
    f.getline(cstring,
      sizeof(cstring));
    cout<<cstring<<endl;
  }
  f.close();
}
```

---

**Einführung Java**

```
import java.awt.*;
import java.awt.event.*;

class Muster extends Panel{
  // hier Referenzen fuer Komponenten
  // (Buttons, Textfields, Panels) vereinbaren
  Button OK;


  public Muster()  {
    // Komponenten erzeugen und zu Oberflaeche
        zusammenbauen,
    // Listener verbinden
```

```
    OK=new Button("OK");
    this.add(OK);
    //addActionListener(...);
  }

  public static void main(String args[])  {
      Frame F=new Frame();
      F.addWindowListener(new WindowAdapter(){public
          void windowClosing(WindowEvent we){System.exit
          (0);}});
      Muster P=new Muster();
      F.add(P);
      F.pack();
      F.setVisible(true);
  }
}
```

**Flow**

```
public class FlowLayoutPanel extends Panel{
  Button b1=new Button("Max");
  Button b2=new Button("Mexi");
public FlowLayoutPanel(){
  setFont(new Font("System", Font.PLAIN, 24));
  setLayout(new FlowLayout());
  add(b1);
  add(b2);
}
public static void main(String args[]){
  FlowLayoutPanel p=new FlowLayoutPanel();
  Frame f=new Frame("FlowLayoutPanel");
  f.add(p);
  f.addWindowListener(new WindowAdapter(){
  public void windowClosing(WindowEvent e){
  System.exit(0);}
  });
  f.setVisible(true);
  f.pack();
  }
}
```

**Border Layout:**

```
public BorderLayoutPanel(){
  setFont(new Font("System", Font.PLAIN, 24));
  setLayout(new BorderLayout());
  add(b1,BorderLayout.NORTH);
  add(b2,BorderLayout.EAST);
  add(b3,BorderLayout.SOUTH);
  //add(new Button("TEST"),BorderLayout.SOUTH);
  add(b4,BorderLayout.WEST);
  add(b5,BorderLayout.CENTER);
}
```

**Grid**

```
public GridLayoutPanel(){
  setFont(new Font("System", Font.PLAIN, 24));
  setLayout(new GridLayout(3,2,5,5));
  add(b1);
  add(b2);
}
```

**Card**

```
CardLayout cards=new CardLayout();
public CardPanel(){
  setFont(new Font("System",Font.PLAIN,22));
  setLayout(cards);
  add(b1); b1.addActionListener(this);
  add(b2); b2.addActionListener(this);
}
public void actionPerformed(ActionEvent e){
cards.next(this);
}
```

Auswahl:

```
public CardPanel(String select){
  setLayout(cards);
  add(b1,"Button 1");
  ...
  cards.show(select,this); // z.B.: "Button 1"
}
public static void main(String args[]){
FlowLayoutPanel p=new CardPanel(args[0]);
...
```

**Eventhandling** (Lösung Member)

```
public class Mouse2 extends Panel{
  int PosX=20,PosY=20;
  String S;
  public Mouse2(String S)  {
    this.S=S;
    setFont(new Font("sanserif",Font.BOLD,24));
    addMouseMotionListener(new myMouseMotionListener());
  }
  // Member Class
  class myMouseMotionListener extends MouseMotionAdapter
        {
    public void mouseDragged(MouseEvent e)    {
      PosX=e.getX(); PosY=e.getY(); repaint();
    }
  }

  public void paint(Graphics g)  {
    g.drawString(S,PosX,PosY);
  }

  public static void main(String args[])  {
    Frame F=new Frame();
    Mouse2 P=new Mouse2(args[0]);
    F.add (P, BorderLayout.CENTER);
    F.setSize(500,200);
    F.setVisible(true);
    F.addWindowListener(new WindowAdapter(){public void
        windowClosing(WindowEvent e){System.exit(0);}});
  }
}
```

**Exception**

```
try
{
int array[]={1,2};
int i=Integer.parseInt(args[0]);
System.out.println("Array["+i+"]="+array[i]);
} catch (IndexOutOfBoundsException e)
{
System.out.println("myException: "+e); e.printStackTrace
    ();
} catch (NumberFormatException n)
{
System.out.println("myException: "+n); n.printStackTrace
    ();
}
```