```
/** Hartmut Helmke 22.12.06 / 13.11.2010
adaptiert von Vincent Godin

Änderungen:
25-03-2011 HHe  Farbe violett korrigiert, war vorher gelb

Das Programm kann verwendet werden, um Farben im Dos-Fenster
darzustellen, wenn man Ausgaben mit cout durchführt.
Beispiel:

#include "DosFarben.h"

int main(){

   printScreenColorOnceVal(cerr, FOREGROUND_GREEN | FOREGROUND_RED |FOREGROUND_INTENSITY, "Fehlerrate:");
   int number=4594;
   printScreenColorOnceVal(cout, FOREGROUND_RED|FOREGROUND_INTENSITY, number);
   cout << " , ";
   printScreenColorOnceRef(cout, FOREGROUND_GREEN|FOREGROUND_INTENSITY, number);
   cout << "!!!\n";

   std::cout << red <<  "hallo" << "\n" << "neu" << endl;
   std::cout << blue <<  "hallo" << "\n" << "neu" << endl;
   std::cout << green <<  "hallo" << "\n" << "neu" << endl;
   std::cout << yellow <<  "hallo" << "\n" << "neu" << endl;
   std::cout << black <<  "hallo" << "\n" << white << "after black" << endl;
   std::cout << white <<  "hallo" << "\n" << "neu" << endl;
   std::cout << color(FOREGROUND_RED | FOREGROUND_BLUE ) <<  "hallo"
      << white << "\n" << "neu" << endl;
   }
*/

#ifndef DosFarben_HEADER
#define DosFarben_HEADER



#include <iostream>

#if  (!defined(_MSC_VER))

static const short RED_SCREEN_COLOR=0;
static const short GREEN_SCREEN_COLOR=0;
static const short BLUE_SCREEN_COLOR=0;

static const short YELLOW_SCREEN_COLOR=0;
static const short BLACK_SCREEN_COLOR=0;
static const short WHITE_SCREEN_COLOR=0;
static const short VIOLETT_SCREEN_COLOR = 0;
static const short GREY_SCREEN_COLOR = 0;


inline std::ostream& red(std::ostream& s) {
   return s;
   }

inline std::ostream& blue(std::ostream& s) {
   return s;
   }

inline std::ostream& green(std::ostream& s) {
   return s;
   }

inline std::ostream& yellow(std::ostream& s) {
   return s;
   }

inline std::ostream& violett(std::ostream& s) {
   return s;
   }

inline std::ostream& white(std::ostream& s) {
   return s;
   }

inline std::ostream& grey(std::ostream& s) {
   return s;
   }

inline std::ostream& black(std::ostream& s) {
   return s;
   }


struct color {
   color(short attribute): m_color(attribute) {};
   short m_color;
   };
template <class _Elem, class _Traits>
std::basic_ostream<_Elem, _Traits>&
operator<<(std::basic_ostream<_Elem, _Traits>& s, const color& ) {
   return s;
   }

inline
short SetAndRememberColor(short new_color)
{
    return new_color;
}

typedef int WORD;
template <typename T>
inline
void printScreenColorOnceVal(std::ostream& s, WORD, T text)
{
   s << text;
}

template <typename T>
inline
void printScreenColorOnceRef(std::ostream& s, WORD, const T& text)
{
   s << text;
}

#else

#include <windows.h>

static const WORD RED_SCREEN_COLOR = FOREGROUND_RED | FOREGROUND_INTENSITY;
static const WORD GREEN_SCREEN_COLOR = FOREGROUND_GREEN | FOREGROUND_INTENSITY;
static const WORD BLUE_SCREEN_COLOR = FOREGROUND_BLUE | FOREGROUND_INTENSITY;

static const WORD YELLOW_SCREEN_COLOR =  FOREGROUND_GREEN | FOREGROUND_RED |FOREGROUND_INTENSITY;
static const WORD BLACK_SCREEN_COLOR = 0;
static const WORD WHITE_SCREEN_COLOR =
   FOREGROUND_BLUE | FOREGROUND_RED |FOREGROUND_GREEN | FOREGROUND_INTENSITY;
static const WORD VIOLETT_SCREEN_COLOR = FOREGROUND_RED | FOREGROUND_BLUE;
static const WORD GREY_SCREEN_COLOR = FOREGROUND_INTENSITY;

inline std::ostream& red(std::ostream& s) {
   HANDLE hStdout = GetStdHandle(STD_OUTPUT_HANDLE);
   SetConsoleTextAttribute(hStdout, RED_SCREEN_COLOR);
   return s;
   }

inline std::ostream& blue(std::ostream& s) {
   HANDLE hStdout = GetStdHandle(STD_OUTPUT_HANDLE);
   SetConsoleTextAttribute(hStdout, BLUE_SCREEN_COLOR);
   return s;
   }

inline std::ostream& green(std::ostream& s) {
   HANDLE hStdout = GetStdHandle(STD_OUTPUT_HANDLE);
   SetConsoleTextAttribute(hStdout, GREEN_SCREEN_COLOR);
   return s;
   }

inline std::ostream& yellow(std::ostream& s) {
   HANDLE hStdout = GetStdHandle(STD_OUTPUT_HANDLE);
   SetConsoleTextAttribute(hStdout, YELLOW_SCREEN_COLOR);
   return s;
   }

inline std::ostream& violett(std::ostream& s) {
   HANDLE hStdout = GetStdHandle(STD_OUTPUT_HANDLE);
   SetConsoleTextAttribute(hStdout, VIOLETT_SCREEN_COLOR);
   return s;
   }

inline std::ostream& white(std::ostream& s) {
   HANDLE hStdout = GetStdHandle(STD_OUTPUT_HANDLE);
   SetConsoleTextAttribute(hStdout, WHITE_SCREEN_COLOR);
   return s;
   }

inline std::ostream& grey(std::ostream& s) {
   HANDLE hStdout = GetStdHandle(STD_OUTPUT_HANDLE);
   SetConsoleTextAttribute(hStdout, GREY_SCREEN_COLOR);
   return s;
   }

inline std::ostream& black(std::ostream& s) {
   HANDLE hStdout = GetStdHandle(STD_OUTPUT_HANDLE);
   SetConsoleTextAttribute(hStdout, BLACK_SCREEN_COLOR);
   return s;
   }


struct color {
   color(WORD attribute): m_color(attribute) {};
   WORD m_color;
   };
template <class _Elem, class _Traits>
std::basic_ostream<_Elem, _Traits>&
operator<<(std::basic_ostream<_Elem, _Traits>& s, const color& c) {
   HANDLE hStdout = GetStdHandle(STD_OUTPUT_HANDLE);
   SetConsoleTextAttribute(hStdout, c.m_color);
   return s;
   }

inline
WORD SetAndRememberColor(WORD new_color)
{
    WORD old_color=0;
    HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);
    CONSOLE_SCREEN_BUFFER_INFO csbi;
    GetConsoleScreenBufferInfo(hStdOut,&csbi);
    old_color = csbi.wAttributes;
    SetConsoleTextAttribute(hStdOut,new_color);
    return old_color;
}

template <typename T>
inline
void printScreenColorOnceVal(std::ostream& s, WORD col, T text)
{
   WORD oldColor = SetAndRememberColor(col);
   s << text;
   s << color(oldColor);
}

template <typename T>
inline
void printScreenColorOnceRef(std::ostream& s, WORD col, const T& text)
{
   WORD oldColor = SetAndRememberColor(col);
   s << text;
   s << color(oldColor);
}
#undef ERROR
#endif
#endif
```