Objektorientierte Systeme 1 - SWB2 & TIB2
Hausaufgabe 1

Aufgabe 1: Länderspezifische Ausgabeeinstellungen

In der Standardkonfiguration im Visual Studio werden Umlaute auf der Konsole als unerwünschte Sonderzeichen ausgegeben. Wenn Sie auch Umlaute korrekt ausgeben wollen, können Sie mit `setlocale(LC_ALL, "")` aus der Bibliothek `<clocale>` die Einstellung der Installationsumgebung für Ihre Programme übernehmen. Kompilieren Sie das nachfolgende Beispiel und verstehen Sie die Ausgabe.

```
#include <clocale>  // für setlocale
#include <ctime>    // für time_t, struct tm, time,
                    // strftime, localtime
#include <iostream>
using namespace std;

int main ()
{
  time_t rawtime;
  struct tm * timeinfo;
  time ( &rawtime );
  timeinfo = localtime ( &rawtime );
  char buffer[80] = { 0 };

  // Derzeitige Lokalitätseinstellungen augeben
  cout << "Locale ist: " << setlocale(LC_ALL,NULL) << endl;
  strftime (buffer,80,"%c",timeinfo);
  cout << "Zeit: " << buffer;
  cout << " Dies ist ein Test: üäöÜÖÄß. Ok?" << endl;

  // Lokalitätseinstellungen der installierten Umgebung aktivieren
  setlocale(LC_ALL, "");
  cout << "Locale ist: " << setlocale(LC_ALL,NULL) << endl;
  strftime (buffer,80,"%c",timeinfo);
  cout << "Zeit: " << buffer;
  cout << " Dies ist ein Test: üäöÜÖÄß. Ok?" << endl;

  // Einzelne Lokalitätseinstellungen ändern
  setlocale(LC_TIME, "kor");
  setlocale(LC_CTYPE, "jpn");
  cout << "Locale ist: " << setlocale(LC_ALL,NULL) << endl;
  strftime (buffer,80,"%c",timeinfo);
  cout << "Zeit: " << buffer;
  cout << " Dies ist ein Test: üäöÜÖÄß. Ok?" << endl;

  return 0;
```

```
}
```

Aufgabe 2: Ausgaben formatieren

Führen Sie die beiden folgenden Programme aus und vergleichen Sie die Ergebnisse. Passen Sie dann Programm 2 so an, dass die Ergebnisse beider Programme identisch sind.

Programm 1:

```
#include <math>
#include <stdio.h>
int basis, exponent;
int main(void) {
  printf("\nBasis = ");
  scanf ("%d", &basis);
  printf("\nExponent = ");
  scanf("%d", &exponent);
  printf ("\nErgebnis: \n");
  printf ("\n%d^%d = %d\n", basis, exponent,
    int(pow(double(basis), double(exponent))));
  return 0;
}
```

Programm 2:

```
#include <cmath>
#include <iostream>
using namespace std;
int basis, exponent;
int main(void) {
  cout << endl << "Basis = ";
  cin >> basis;
  cout << "Exponent = ";
  cin >> exponent;
  cout << endl << "Ergebnis: ";
  cout << basis << exponent << " = "
    << pow(basis, exponent);
  return 0;
}
```

Aufgabe 3: Daten einlesen und ausgeben, for each-Schleifen

Schreiben Sie ein Programm, bei dem der Nutzer fünf ganze Zahlen eingibt. Diese Zahlen werden in einem Array gespeichert. Programmieren Sie dann die folgenden Aktionen:

1. Geben Sie alle Zahlen im Array aus.
2. Erhöhen Sie jedes Element um 1 und geben Sie das Array aus.
3. Berechnen Sie den Durchschnitt der Zahlen im Array und geben Sie den Durchschnitt aus.

4. Berechnen Sie die Differenz zweier im Array benachbart gespeicherter Zahlen und geben Sie die Werte aus.

Nutzen Sie soweit möglich for each-Schleifen (range-based loops). Für die Ein- und Ausgabe nutzen Sie die Streams `cin` und `cout`.

Aufgabe 4: Aufzählungstypen und funktionales Casting

Das folgende Programm enthält Fehler und kann nicht erfolgreich kompiliert werden, weder mit dem C-Compiler noch mit dem C++-Compiler. Die Fehler sind aber je nach Compiler unterschiedlich. Kompilieren Sie das Programm sowohl mit dem C-Compiler als auch mit dem C++-Compiler und finden Sie die Fehler. Markieren Sie die Fehler am Ende der einzelnen Zeilen und erklären Sie sie. Schaffen Sie jeweils für den C- und den C++-Compiler eine kompilierbare Version des Programms, indem Sie die fehlerhaften Zeilen auskommentieren. Die Versionen sollen maximal in dem Sinne sein, dass so wenig Zeilen wie möglich auskommentiert werden.

```
// Vergleich von Deklaration, Enumeration und Typumwandlung
// in C und C++

// Fehler                                                  C     |  C++
enum enumeration { 1st, 2nd, 3rd, 4th };      // Z1             |
enum marks { '1', '2', '3', '4', '5', '6' }; // Z2             |
enum class1 { DIETER, HANS, PETER, KAI };     // Z3             |
enum class2 { ANDREA, KAI, SABINE, ZOE };     // Z4             |
enum sports { FUSSBALL, BASKETBALL, HOCKEY };// Z5             |

int main(void)                                // Z7             |
{
    int i;                                    // Z8             |
    sports mysport;                           // Z9             |
    enum sports mysport;                      // Z10            |
    mysport = FUSSBALL;                       // Z11            |
    mysport = FUSSBALL + 1;                   // Z12            |
    mysport = 1;                              // Z13            |
    mysport = 10;                             // Z14            |
    mysport = mysport + 1;                    // Z15            |
    mysport = sports(mysport + 1);            // Z16            |
    float f;                                  // Z17            |
    for (i = FUSSBALL; i <= HOCKEY; i++)      // Z18            |
        mysport = sports(i);                  // Z19            |
    for (int k = FUSSBALL; k <= HOCKEY; k++) // Z20            |
        mysport = sports(k);                  // Z21            |
    return 0;                                 // Z22            |
}
```

Aufgabe 5: Aufzählungstypen mit `enum class`

Nehmen Sie das Originalprogramm von der vorherigen Aufgabe und ändern Sie die Aufzählungstypen von **`enum`** in **`enum class`**. Was muss dann noch beim Programm angepasst werden, so dass es richtig funktioniert? Kommentieren Sie so wenige Zeilen wie möglich aus und führen sie die notwendigen Änderungen durch.

Aufgabe 6: Speicherbedarf von Referenzen

Schauen Sie sich das folgende Programm an und beschreiben Sie, was Sie als Ausgabe erwarten. Führen Sie dann das Programm aus, um Ihre Vorhersage zu überprüfen.

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

int main(void)
{
    int i = 1;
    int & j = i;
    int * ptr;
    ptr = &i;
    cout << "Groesse von i: " << sizeof(i) << endl;
    cout << "Groesse von j: " << sizeof(j) << endl;
    cout << "Groesse von ptr: " << sizeof(ptr) << endl;
    cout << boolalpha;
    cout << "Die Adressen von i und j sind gleich: ";
    cout << (&i == &j) << endl;
    cout << "Die Adressen von j und *ptr sind gleich: ";
    cout << (&j == &(*ptr)) << endl;
    cout << "Die Adressen von j und ptr sind gleich: ";
    cout << (int(&j) == int(&ptr)) << endl;
    return 0;
}
```

Aufgabe 7: Referenzen als Rückgabewert

Das nachfolgende Programm hat einen Fehler. Erklären Sie diesen und korrigieren Sie dann das Programm, so dass es korrekt arbeitet. Dazu müssen Sie lediglich ein Zeichen einfügen.

```
#include <iostream>
#include <ctime>
using namespace std;
unsigned int groessergleich50 = 0;
unsigned int kleiner50 = 0;

unsigned int teste( unsigned int wert )
{
  if ( wert >= 50 )
    return groessergleich50;
  else
    return kleiner50;
}
```

```
int main( void )
{
  int i=0;
  srand( (unsigned)time( NULL ) );
    // Initialisiert den Zufallsgenerator
  while ( i < 100000 )
  {
    (teste( rand() % 100 ) )++;
    i++;
  }
  cout << "Ergebnis der Funktion teste:" << endl;
  cout << "Von "<< i <<" Zahlen zwischen 0 und 99 waren " << endl;
  cout << kleiner50 << " Zahlen zwischen 0 und 49" << endl;
  cout << groessergleich50 << " Zahlen zwischen 50 und 99" << endl;
  return 0;
}
```

Aufgabe 8: Referenzen als lvalues und rvalues

Analysieren Sie das nachfolgende Programm. Überlegen Sie sich, welche Werte im Array s nach Ausführung der markierten Zeilen („Zustand ...“) stehen. Verifizieren Sie Ihre Analyse mit einem Programmdurchlauf, in dem Sie das Array ausgeben.

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

char s[4];

char & select(int i, const char & c)
{
    s[i]=c;
    return s[i];
}

int main(void)
{
    select(0, 'a');
    select(1, 'b');
    select(2, 'c');              // s[0]  s[1]  s[2]  s[3]
    select(3, 'd');              // Zustand 1:
    select(0, 'A');              // Zustand 2:
    select(0, 'A') = '0';        // Zustand 3:
    select(1, 'B')
        = select(2, 'C');        // Zustand 4:
    select(1, select(2, '2'))
        = select(3, '3');        // Zustand 5:
    return 0;
}
```

Aufgabe 9: Überladene Funktionen mit Defaultparametern

Analysieren Sie das nachfolgende Programm und markieren Sie die Zeilen, die Fehler verursachen. Kommentieren Sie dann die minimale Anzahl von Funktionsdefinitionen aus, so dass Sie das Programm erfolgreich kompilieren können. Kommentieren Sie keine Zeile im Hauptprogramm aus. Nennen Sie die Funktionen, die im Hauptprogramm aufgerufen werden, und erklären Sie Ihre Überlegungen. Verifizieren Sie Ihre Überlegungen durch einen Programmdurchlauf.

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

int f(int i) {
    cout << "Funktion 1: i=" << i << endl;
    return 0; }

void f(int i) { cout << "Funktion 2: i=" << i << endl; }

char f(int i) {
    cout << "Funktion 3: i=" << i << endl;
    return 'a'; }

int f(int & i) {
    cout << "Funktion 4: i=" << i << endl;
    return 1; }

int f(int i, int k) {
    cout << "Funktion 5: i=" << i << " k=" << k << endl;
    return 1; }


int f(char c, int k) {
    cout << "Funktion 6: c=" << c << " k=" << k << endl;
    return 1; }

int f(int i, int k=0) {
    cout << "Funktion 7: i=" << i << " k=" << k << endl;
    return 1; }

int f(double d, int i, char c='a') {
    cout<<"Funktion 8: d="<<d<<" i="<<i<<" c="<<c<<endl;
    return 1; }

int f(double d, double e=1.1, int i=0) {
    cout<<"Funktion 9: d="<<d<<" e="<<e<<" i="<<i<<endl;
    return 1; }

int f(double d, double e=1.1, char c) {
    cout<<"Funktion 10: d="<<d<<" e="<<e<<" c="<<c<< endl;
    return 1; }
```

```
int f(char c, char d, int i=2, char e='c') {
    cout<<"Funktion 11: c="<<c<<" d="<<d<<" i="<<i<<" e="<<e<<endl;
    return 1;
}

int main(void)
{
    int i = 1;
    char c = 'a';
    f(1);                 // Ausgefuehrte Funktion:
    f(c);                 // Ausgefuehrte Funktion:
    f(i);                 // Ausgefuehrte Funktion:
    c = f('a');           // Ausgefuehrte Funktion:
    c = 'a';
    f(1, 2);              // Ausgefuehrte Funktion:
    f(c, i);              // Ausgefuehrte Funktion:
    f(1.0, 2.0);          // Ausgefuehrte Funktion:
    f(1.0, 2.0, 1);       // Ausgefuehrte Funktion:
    f(1.0, 1, 'a');       // Ausgefuehrte Funktion:
    f(1.0, 1);            // Ausgefuehrte Funktion:
    f(1.0, double(i));    // Ausgefuehrte Funktion:
    f(c, c, i);           // Ausgefuehrte Funktion:
    f(c, i);              // Ausgefuehrte Funktion:
    return 0;
}
```