Aufgabe 1 (Strings und Vektoren). **(4+4 Punkte)**

a) Geben Sie eine Funktion an, die als Argument einen String erwartet (`std::string`) und die von diesem String die Buchstaben mit geradem Index auf der Konsole ausgibt.

Beispiel: Für den String "`abcdef`" gibt die Funktion "`ace`" aus.

Lösung:

```
void printeven(std::string s) {
    for (int i=1;i<s.size();i+=2)
        std::cout << s[i];
}
```

b) Geben sie eine Funktion an, die als Argument einen Vektor aus ganzen Zahlen erwartet und die das Produkt der Elemente zurückliefert.

Lösung:

```
int produkt(std::vector<int> v) {
    int res = 1;
    for (int el:v) {
        res *= el;
    }
    return res;
}
```

Aufgabe 2 (Fehlersuche). **(6+6 Punkte)**

Betrachten Sie das folgende Programm:

```
include<iostream>
#include<string>

void foo(std::string s) {
    i=0;
    while s[i] != 'a' {
        i++;
    }
    while (i != 0) {
        std::cout << s[i] << '\n';
        i--
    }


int main() (
    foo("Dieses Programm macht dubiose Dinge!");
    return 42;
}
```

a) Das Programm hat 6 Fehler. Markieren Sie die Fehler.
Für jede falsche Markierung wird Ihnen ein Punkt abgezogen.

b) Geben Sie jeweils an, was falsch ist bzw. wie der Fehler korrigiert werden kann.

Hinweis: In jeder Zeile gibt es höchstens einen Fehler. **Ein Fehler liegt nur vor, wenn das Programm nicht compiliert werden kann.** Wenn eine Zeile nur eine Warnung produziert oder das Programm sich unerwartet verhält, liegt kein Fehler vor.

Lösung:

```
#include<iostream>                                    // Fehler
#include<string>

void foo(std::string s) {
    int i=0;                                          // Fehler
    while (s[i] != 'a') {                             // Fehler
        i++;
    }
    while (i != 0) {
        std::cout << s[i] << '\n';
        i--;                                          // Fehler
    }
}                                                     // Fehler

int main() {                                          // Fehler
```

```
    foo("Dieses Programm macht dubiose Dinge!");
    return 42;
}

/* Fehler:
 * - Zeile 1: '#' fehlt.
 * - Zeile 5: Deklaration von 'i' fehlt.
 * - Zeile 6: Klammern bei While-Bedingung fehlen.
 * - Zeile 11: Semikolon fehlt.
 * - Zeile 13: Schließende geschweifte Klammer fehlt.
 * - Zeile 15: Runde statt geschweifter Klammer.
 */
```

Aufgabe 3 (Überladen von Funktionen). **(5+2 Punkte)**

a) Überladen Sie den Operator `+` für den Datentyp `vector<int>`. D.h. für zwei `int`-Vektoren `v1` und `v2` soll `v1 + v2` die Verkettung von `v1` und `v2` sein.

Lösung:

```
std::vector<int> operator+(std::vector<int> v1, std::vector<int> v2) {
    for (int el:v2) v1.push_back(el);
    return v1;
}
```

b) Überladen Sie die folgende Funktion für den Datentyp `vector<int>`.

```
std::string verdoppeln(std::string s) { return s + s; }
```

Lösung:

```
std::vector<int> verdoppeln(std::vector<int> v) { return v + v; }
```

Hier wird angenommen, dass der Operator `+` wie in Aufgabenteil a) überladen ist.

Aufgabe 4 (Klassen). **(3+8 Punkte)**

a) Implementieren Sie den Konstruktor der folgenden Klasse, so dass er die Attribute der Klasse initialisiert.

```
class adresse {
private:
    std::string name;
    std::string strasse;
    std::string ort;
public:
    adresse(std::string n, std::string s, std::string o);
};
```

Lösung:

```
adresse::adresse(std::string n, std::string s, std::string o) :
        name{n}, ort{o}, strasse{s} {}
```

b) Definieren Sie eine Klasse `adressliste`, die eine Reihe von Adressen speichert. Die Klasse soll die folgenden Methoden haben:

- Eine Methode zum Hinzufügen einer neuen Adresse.
- Eine Methode, die als Argument einen Namen erwartet und die dazugehörige Adresse auf der Konsole ausgibt. Ist der Name nicht vorhanden, soll eine Fehlermeldung ausgegeben werden.

Erläutern Sie dabei ggf., welche Änderungen an `adresse` notwendig sind.

Lösung:

```
class adressliste {
private:
    std::vector<adresse> liste;

public:
    void hinzufuegen(adresse a) { liste.push_back(a); }
    void ausgeben(std::string name) {
        // Ansatz: 'name' in Liste suchen.
        // Falls gefunden, Adresse ausgeben und vorzeitig abbrechen.
        for (adresse a:liste) {
            if (a.get_name() == name) {
                std::cout << a.get_name() << "\n"
                          << a.get_strasse() << "\n"
                          << a.get_ort() << "\n";
                return;
            }
        }
```

```
            // Hier kommen wir nur an, wenn 'name' nicht gefunden wurde.
            std::cout << "Keine Adresse zu \"" << name
                      << "\" gefunden.\n";
        }
    };
```

In der Klasse `adresse` werden noch Getter-Methoden für die Attribute gebraucht:

```
    std::string get_name() { return name; }
    std::string get_strasse() { return strasse; }
    std::string get_ort() { return ort; }
```

Aufgabe 5 (Programmverständnis). **(8 Punkte)**

Geben Sie die Ausgabe des folgenden Programms an:

```
#include<iostream>

int foo(int x, int y) {
    std::cout << x;
    return x + y;
}

void bar(int & k) {
    int y = k;
    for (char b=3;b>=-1;b--) {
        k = foo(y,b);
    }
}

int main() {
    int x = 1;
    while (x != 0) bar(x);
    return 0;
}
```

Lösung:
Die Ausgabe ist `1 1 1 1 1`.

Genauere Erklärung

```
#include<iostream>

/* foo():
 * - Es wird 'y' aus 'bar()' ausgegeben, also 1.
 * - Beim letzten Mal wird dann 0 zurückgegeben.
 *   Vorherige Werte spielen keine Rolle (s.u.).
 */
int foo(int x, int y) {
    std::cout << x;
    return x + y;
}

/* bar()
 * - 'y' hat am Anfang den Wert k (== 1) und wird nicht mehr verändert.
 * - 5 Schleifendurchläufe, der letzte mit 'b == -1'.
 * - nur der letzte Durchlauf ist für den finalen Wert
 *   von 'k' relevant: Am Ende gilt 'k == 0'
 */
void bar(int & k) {
    int y = k;
    for (char b=3;b>=-1;b--) {
```

```
        k = foo(y,b);
    }
}

/* main():
 * - Der Aufruf von 'bar(x)' verändert 'x'.
 * - Nach einem Aufruf ist 'x == 0' (s.o.).
 */
int main() {
    int x = 1;
    while (x != 0) bar(x);
    return 0;
}

/* Zusammenfassung:
 * - Die While-Schleife läuft einmal durch, 'bar()' wird einmal aufgerufen.
 * - 'bar()' durchläuft die For-Schleife 5 Mal, Am Ende ist 'k == x == 0'.
 * - Jeder Aufruf von 'foo()' gibt den Wert von 'y' aus 'bar()' aus. Das ist i
 * - Also wird fünf Mal die 1 ausgegeben.
 */
```

Aufgabe 6 (Referenzen, Polymorphie). **(8 Punkte)**

Geben Sie die Ausgabe des folgenden Programms an:

```
#include<iostream>
#include<string>

class name {
protected :
    std::string vorname = "Max";
    std::string nachname = "Mustermann";
public :
    void print() { std::cout << vorname << " " << nachname << "\n"; }
    virtual int length() { return vorname.size() + nachname.size() + 1; }
};

class firmenname : public name {
public :
    firmenname() { nachname = "AG"; }
    void print() { std::cout << nachname << "\n"; }
    virtual int length() { return nachname.size(); }
};


int main() {
    name n;
    firmenname f;
    name &rn = n;
    name * pf = &f;

    n.print();
    std::cout << n.length() << "\n";
    f.print();
    std::cout << f.length() << "\n";

    rn.print();
    std::cout << rn.length() << "\n";
    pf->print();
    std::cout << pf->length() << "\n";
}
```

Lösung:

```
Max Mustermann
14
AG
2
Max Mustermann
14
```

Max AG
2

Aufgabe 7 (Templates). **(6 Punkte)**

Geben Sie eine parametrisierte Funktion `replace` an, die einen `vector` `v` und zwei dazu passende Elemente `x` und `y` erwartet. Die Funktion soll einen Vektor zurückliefern, in dem jedes Vorkommen von `x` durch `y` ersetzt ist.

Die Funktion soll mit jedem beliebigen `vector` funktionieren.

Lösung:

```
template<class T>
std::vector<T> replace(std::vector<T> v,T x, T y) {
    for (T & el:v) el = (el==x?y:x);
    return v;
}
```