Informatik I (D-ITET)
Übungsstunde 9, 21.11.2016

Hossein Shafagh
shafagh@inf.ethz.ch

Self-Assessment next Week!
Problem 8.1. Vector and matrix multiplication

```cpp
int main() {
    char ltype;
    std::cout << "enter left operand type:\n";
    std::cin >> ltype;
    char rtype;
    std::cout << "enter right operand type:\n";
    std::cin >> rtype;

    if (ltype == 'v' && rtype == 'v') {
        vector_vector();
    } else if (ltype == 'm' && rtype == 'v') {
        matrix_vector();
    } else if (ltype == 'm' && rtype == 'm') {
        matrix_matrix();
    } else {
        std::cout << "unsupported operation\n";
    }

    return 0;
}
```
Problem 8.1. Vector and matrix multiplication

- Einlesen

```c
void matrix_vector() {
    int matrix[3][3];
    int vector[3];
    int res[3];

    input_matrix(matrix);
    input_vector(vector);

    matrix_vector_product(res, matrix, vector);

    output_vector(res);
}
```
Problem 8.1. Vector and matrix multiplication

- Einlesen

// POST: matrix filled with values from std input
void input_matrix(int (&matrix)[3][3]) {
    for (int row = 0; row != 3; ++row) {
        for (int col = 0; col != 3; ++col)
            std::cin >> matrix[row][col];
    }
}
Problem 8.1. Vector and matrix multiplication

// POST: result contains the cross product of left and right
void cross_product(int (&res)[3], const int (&left)[3], const int (&right)[3]) {
}

// POST: result contains the matrix-vector product of left and right
void matrix_vector_product(int (&res)[3], const int (&left)[3][3], const int (&right)[3]) {
    // row of left array
    for (int row = 0; row != 3; ++row) {
        int sum = 0;
        // column of right array
        for (int col = 0; col != 3; ++col) {
            // column of left array = row of right array
            sum += left[row][col] * right[col];
        }
        res[row] = sum;
    }
}
Problem 8.1. Vector and matrix multiplication

// POST: result contains the matrix product of left and right
void matrix_product(int (&res)[3][3], const int (&left)[3][3], const int (&right)[3][3]) {
    // row of left array
    for (int row = 0; row != 3; ++row) {
        // column of right array
        for (int col = 0; col != 3; ++col) {
            // column of left array = row of right array
            int sum = 0;
            for (int i = 0; i != 3; ++i) {
                sum += left[row][i] * right[i][col];
            }
            res[row][col] = sum;
        }
    }
}
Problem 8.2. Decode binary NZZ front page

```cpp
#include <iostream>
#include <fstream>

int main() {
    std::string filename;
    std::cin >> filename;

    std::ifstream in(filename.c_str());
    std::string value;

    while(in >> value) {
        unsigned char c;
        bool success = decode_to_char(c, value);
        if (!success) {
            std::cout << "input file has incorrect format\n";
            break;
        }
        std::cout << c;
    }
    return 0;
}
```
Problem 8.2. Decode binary NZZ front page

// PRE: Input string value containing exactly 8 characters. Allowed characters are '0' and '1'.
// POST: Returns true if the string value could be converted to a character, otherwise false.
// If return value is true then res contains the character corresponding to the input string
// value, otherwise res is undefined.

bool decode_to_char(unsigned char& res, const std::string& value) {
    if (value.length() != 8) {
        return false;
    }
    res = 0;
    for(int i = 0; i < 8; ++i) {
        res *= 2;
        if (value[i] == '1') {
            res += 1;
        } else if (value[i] != '0') {
            return false;
        }
    }
    return true;
}
include <iostream>
using namespace std;

int main () {
    int var1;
    char var2[10];

    cout << "Address of var1 variable: ";
    cout << &var1 << endl; // Address of var1 variable: 0xbfebd5c0

    cout << "Address of var2 variable: ";
    cout << &var2 << endl; // Address of var2 variable: 0xbfebd5b6

    return 0;
}
1 Pointers

int *ip; // pointer to an integer
double *dp; // pointer to a double
float *fp; // pointer to a float
char *ch // pointer to character
1 Pointers

```c
int main () {
    int var = 20;  // actual variable declaration.
    int *ip;      // pointer variable

    ip = &var;    // store address of var in pointer variable

    cout << "Value of var variable: ";
    cout << var << endl;    // Value of var variable: 20

    // print the address stored in ip pointer variable
    cout << "Address stored in ip variable: ";
    cout << ip << endl;     // Address stored in ip variable: 0xbfc601ac

    // access the value at the address available in pointer
    cout << "Value of *ip variable: ";
    cout << *ip << endl;    // Value of *ip variable: 20
}
```
1 Pointers

```c
int *pc, c;
c = 5;

pc = &c;

pc = 0xFFF5F880c
pc = 5
*pc = 5

*pc = 2;

&c = 0xFFF5F880c
c = 2
```
1 Pointers

- Common mistakes when working with pointers

```c
int c, *pc;

pc=c;       // Wrong! pc is address whereas, c is not an address.

*pc=&c;     // Wrong! *pc is the value pointed by address whereas, c is an address.

pc=&c;      // Correct! pc is an address and, pc is also an address. */

*pc=c;      // Correct! *pc is the value pointed by address and, c is also a value. */
```
1 Pointers

- Pointer Program
2 Iterators for Vectors

- Vectors have a similar concept as pointers, called iterators.
- Similar to array and pointers
- .begin() and .end() on a vector to get the iterators
- the & operator unnecessary for vectors

```cpp
#include <vector>

// (for reference: the corresponding pointer example)
// int array[] = {8,3,1,4,6,9};
// for (int* p = array; p != array + 6; ++p)
//   std::cout << *p;
std::vector<int> vec = {8,3,1,4,6,9}; // C++11 syntax!

for (std::vector<int>::iterator it = vec.begin(); it < vec.end(); ++it)
  std::cout << *it;
```
3 Typedef

- gives a new name to an already existing type
- More readable code!
- Easier to maintain!

```cpp
std::vector<int> vec = {8,3,1,4,6,9};
for (std::vector<int>::iterator it = vec.begin(); it < vec.end(); ++it)
    std::cout << *it;

typedef std::vector<int> intvec;
typedef std::vector<int>::iterator intvecit;

intvec vec = {7,3,5,2,7,9};
for (intvecit it = vec.begin(); it < vec.end(); ++it)
    std::cout << *it;
```
4 const Pointers

```c
int x = 7;
int y = 5;

const int* i = &x;
++(*i); // not compiling, value stored in x cannot be changed via pointer access
i = &y; // works

int* const j = &x;
++(*j); // works
j = &y; // does not compile, address stored in j is const

const int* const k = &x; // one can also do both
++(*k); // not compiling, value stored in x cannot be changed via pointer access
k = &y; // does not compile, address stored in k is const
```
4 const Pointers

- It is not possible to have a non-\texttt{const}-pointer pointing to a \texttt{const}-target:

\begin{verbatim}
const int a = 5;
int* a_ptr = &a; // Error: must be \texttt{const int* a_ptr = a;}
\end{verbatim}
4 const Pointers
5 Recursion

- termination condition

```c
double power(const double x, const unsigned int n) {
    if (n == 1)
        return x;
    return x*power(x, n-1); // x*x*x ... x; n times!
}
```

```c
double power(const double x, const unsigned int n) {
    if (n == 1)
        return x;
    double temp = power(x,n/2);
    return temp*temp;
}
```
5 Recursion

double power(const double x, const unsigned int n) {
    if (n == 1) {
        return x;
    } else if (n % 2 == 0) {
        double temp = power(x, n/2);
        return temp*temp;
    } else {
        return x*power(x, n-1);
    }
}
Problem 9.1. Understanding Pointers

Problem 9.2. Sorting
- integer array
- pointers:
  - void input(int* const begin, int* const end)
- several functions: input, output, swap, min, max, sort

Problem 9.3. Lexicographic comparison
- e.g., a = bicycle, b = bike: a < b, because a[2] < b[2]
- e.g., web < website
- Iterators
- lexicographic_compare
- largeststring (std::vector<std::string> strings(n);)
Problem 9.1. Understanding Pointers

What does the following code output?

a)
```cpp
int a[] = {9, 8, -5};
int* p = &a[2];
std::cout << (p - a) << "\n";
→ 2
```

b)
```cpp
int a[] = {5, 6, 2, 3, 1, 4, 0};
int* p = a;
do {
    std::cout << *p << " ";
    p = a + *p;
} while (p != a);
→ 5 4 1 6 0
```

c)
```cpp
unsigned int a[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
for (unsigned int* p = a; p < a + 10; ++p)
    std::cout << *p << " ";
→ 0 1 2 3 4 5 6 7 8 9
```

d)
```cpp
char a[] = {'K', 'h', 'o', 'o', 'r'};
for (char* it = a; it < a + 5; ++it)
    *it = *it - 3;
for (char* it = a; it < a + 5; ++it)
    std::cout << *it;
→ Hello
```