Exercise Session

Week 08

Adel Gavranović
agavranovic@student.ethz.ch

Overview

▸ polybox for session material ▸ mail to TA

Today's Topics

Introduction

Multidimensional Vectors

Recursion

Comments on last [code]expert Exercises

No matter how many points you received, always check the Master Solution and study it. You'll often find better implementations or neat tricks for the next exercises.

Questions or Comments re: Exercises?

General Comments

- don’t gather in the ETH anymore
- we’ll do a some ”longer” exercises today in ”breakout rooms”

Learning Objectives Checklist

Now I...

- ☐ understand recursion and why it is so useful in computer science
- ☐ can write a program that manipulates multidimensional vectors in C++
- ☐ can write a recursive algorithm for the problem ”Towers of Hanoi”
- ☐ can design and write C++ programs, that tries out all possible solutions of a given problem recursively

What are Multidimensional Vectors?

Multidimensional Vectors are Matrices

Exercise ”Matrix Transpose”

- Open ”Matrix Transpose” on [code]expert

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^{\mathsf{T}} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

- Think about how to solve the problem with pen and paper
- ```
  using irow = std::vector<int>;
  using imatrix = std::vector<irow>;
  ```
- Get into groups of two (in breakout rooms) and try to write a solution
- Share your code and discuss mistakes, inefficiencies and structure
  ```

Solution "Matrix Transpose"

```
// POST: Returns transpose of input matrix
imatrix transpose_matrix(const imatrix &matrix){
  unsigned int r, c;
  r = get_rows(matrix);
  c = get_cols(matrix);
  imatrix transposed_matrix;
  for(unsigned int col_index = 0; col_index < c;
      col_index++){
    irow row;
    for(unsigned int row_index = 0; row_index < r;
        row_index++){
      row.push_back(matrix.at(row_index).at(col_index));
    }
    transposed_matrix.push_back(row);
  }
  return transposed_matrix;
}
```

Questions?

What is Recursion?

Recursion is often useful for solving problems by using the *divide and conquer* approach:

1. If we need to solve the problem for n, find a way to split it into smalles pieces: $k_0, k_1, \ldots, k_m \quad (\forall 0 \leq i \leq m : k_i < n)$
2. Solve each piece k_i seperately (maybe dividing it further)
3. Combine the solution to each piece k_i to the solution for the whole problem

Exercise ”Towers of Hanoi”

:: open TowersOfHanoi.eng.pdf ::

Exercise ”Sequence Permutations”

- Open ”Sequence Permutations” on [code]expert
- Think about how to solve the problem with pen and paper
- Get into groups of two (in breakout rooms) and try to write a solution
- You should use functions and a recursive approach
- Warning: this one is *not easy*
- Share your code and discuss mistakes, inefficiencies and structure

Solution "Sequence Permutations"

```
int main () {
  std::vector<int> sequence;

  // Read input.
  int num;
  std::cin >> num;
  while (num != -1) {
    sequence.push_back(num);
    std::cin >> num;
  }

  // A vector for tracking the current permutations.
  std::vector<int> permutation;

  permute(sequence, permutation);

  return 0;
}
```

Solution "Sequence Permutations"

```
// POST: Returns true iff element is in the permutation.
bool is_used(int element, std::vector<int>& permutation) {
  for (unsigned int i = 0; i < permutation.size(); ++i) {
    if (permutation.at(i) == element) {
      return true;
    }
  }
  return false;
}

// POST: Print the permutation to standard output.
void print_permutation(std::vector<int>& permutation) {
  for (unsigned int i = 0; i < permutation.size(); ++i) {
    std::cout << permutation.at(i) << " ";
  }
  std::cout << std::endl;
}
```

Solution "Sequence Permutations"

```
// POST: Prints out all possible permutations of the
    given sequence.
void permute(std::vector<int> sequence, std::vector<int>
    permutation) {
  if (sequence.size() == permutation.size()) {
    // We have a full permutation, just output it.
    print_permutation(permutation);
  } else {
    // Try all unused elements of the sequence.
    for (unsigned int i = 0; i < sequence.size(); ++i) {
      int element = sequence.at(i);
      if (!is_used(element, permutation)) {
        permutation.push_back(element);
        permute(sequence, permutation);
        permutation.pop_back();
      }
    }
  }
}
```

Questions?