ETH zürich

Exercise Session W08

Computer Science (CSE & CBB & Statistics) – AS 23

Overview

Today's Agenda

Follow-up
Objectives
Multidimensional Vectors
Recursion
Outro

n.ethz.ch/~agavranovic

1. Follow-up

Follow-up

- Thank you all for the overwhelmingly positive Feedback!
- If there's still something you want to tell me, feel free to send me an email (with a throwaway address if you want to stay anonymous)

Questions?

2. Objectives

Objectives

- ☐ be able to write programs using multidimensional vectors
- ☐ be able to understand and write programs using recursion

3. Multidimensional Vectors

What are Multidimensional Vectors?

Multidimensional vectors are Matrices[1]

[1] they're actually vectors of vectors!

Exercise "Matrix Transpose"

- Open "Matrix Transpose" on **code** expert

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

- Think about how you would approach the problem with pen and paper
- Simplification of the syntax:

```
using irow = std::vector<int>;
using imatrix = std::vector<irow>;
```

- Implement a solution (optionally in groups)

Solution to "Matrix Transpose"

```
imatrix transpose_matrix(const imatrix &matrix){
  unsigned int r, c;
  r = get_rows(matrix);        // number of rows
  c = get_cols(matrix);        // number of columns
  imatrix transposed_matrix;   // init' transp. matrix
  for(unsigned int col_i = 0; col_i < c; col_i++){
    irow row;                  // init' transp. row
    // entry-wise add transp. row to transp. matrix
    for(unsigned int row_i = 0; row_i < r; row_i++){
      row.push_back(matrix.at(row_i).at(col_i));
    }
    transposed_matrix.push_back(row);
  }
  return transposed_matrix;
}
```

Questions?

4. Recursion

What is Recursion?

Recursion

often helpful when solving problems using the *divide and conquer-approach*

We want to solve a problem for n

1. Find a way to split the problem into smaller subproblems: $k_0, k_1, \ldots, k_m \quad (\forall 0 \leq i \leq m : k_i < n)$
2. Solve every k_i independently (perhaps by subdividing further)
3. Construct the solution to the problem from the solutions to the subproblems

Example of Recursion

We want to write a function with the following PRE and POSTs

```
// PRE:   a positive integer n
//
// POST:  returns the n-th number of a series x_n, defined as
//        x_n = 2,                      for n = 1
//        x_n = 1,                      for n = 2
//        x_n = x_(n-1) + x_(n-2),      for n > 2
//
// Example:
//  * n == 1 ~~> 2
//  * n == 2 ~~> 1
//  * n == 3 ~~> 3
```

Example of Recursion

```
// PRE:   a positive integer n
//
// POST:  returns the n-th number of a serie x_n, defined as
//        x_n = 2,                    for n = 1
//        x_n = 1,                    for n = 2
//        x_n = x_(n-1) + x_(n-2),    for n > 2

unsigned int compute_element(unsigned int n) {
  if (n == 1) return 2;
  else if (n == 2) return 1;
  else return compute_element(n-1) + compute_element(n-2);
}
```

Video Recommendations

Especially try to follow the concept of the *Recursive Leap of Faith*. It is comparable to the induction hypothesis in an induction proof in maths.

Videos on recursion

- ▸ Towers of Hanoi: A Complete Recursive Visualization
- ▸ 5 Simple Steps for Solving Any Recursive Problem

Exercise "Partial Sum"

Task

Write a function that

1. Computes the sum of all natural numbers below (and equal to) n using recursion and returns this value
2. Outputs all the added terms in ascending order (from 0 to n to the console in the same recursive function)

Exercise "Partial Sum"

- Open "Partial Sum" on **code** expert
- Think about how you would approach the problem with pen and paper
- Implement a (recursive) solution (optionally in groups)

Solution to "Partial Sum"

```
unsigned int partial_sum(const unsigned int n) {
  if (n == 0){
    return 0;
  } else {
    // print descending
    unsigned int partial = partial_sum(n - 1);

    // print ascending
    std::cout << n << std::endl;

    return n + partial;
  }
}
```

Solution to "Partial Sum"

```
int main() {
  std::cout << "n = ";

  unsigned int n;
  std::cin >> n;

  std::cout << partial_sum(n) << std::endl;

  return 0;
}
```

Questions?

Exercise "Power Function"

Question

How many recursive calls does the following function need to compute x^7?

```
unsigned int power(const unsigned int x, const unsigned int n) {

    if (n == 0){
        return 1;
    } else if (n ==1) {
        return x;
    }

    return x * power(x, n - 1);
}
```

Answer:7

Exercise "Power Function"

Task

Write a function that requires significantly less recursive calls for larger n.
How many recursive calls does your implementation require?

Exercise "Power Function"

- Open "Power Function" on **code** expert
- Think about how you would approach the problem with pen and paper
- Implement a (recursive) solution (optionally in groups)
- Hint: This task is a generalization of the task "Multiply with 29" from the first week

Solution to "Power Function"

```
// POST: result == x^n
unsigned int power (const unsigned int x, const unsigned int n) {
  if(n == 0) {
    return 1;
  } else if(n == 1) {
    return x;
  } else if(n % 2 == 0) {
    int temp = power(x, n/2);
    return temp * temp;
  } else {
    return x * power(x, n-1);
  }
}
```

Questions?

5. Outro

General things regarding **code** expert

E8:T1: "Vector and matrix operations"

- The task can seem very daunting. Keep an overview over all the different possible cases (perhaps using sketches) and try to implement separate functions for the operations.
- Use `using` to make the program clearer
- Don’t forget `// comments` &references, and `const`!

General Questions?

Till next time!

Cheers!