Submitted by Sonja Jolanda Klein

Handout 2
C++ Programming
Deadline is
October 11

1.

1) Write a function that computes the value of the binomial coefficient $C(n,r)\ or\ \binom{n}{r}$

$$C(n,r) = \frac{n!}{(n-r)!\,r!}$$

$$n! = n * (n-1) * (n-2) * \cdots * 1$$

2) Embed your function into a little program that reads two integers **`n`** and **`r`** from **`std::cin`** and writes the value of the binomial coefficient to **`std::cout`**

Exercise 1.1 and 1.2:

```cpp
#include <iostream>

int factorial(int n) {
    return (n == 1) ? n : n * factorial(n - 1);
}

int calc_binomical_coefficient(int n, int r) {
    return factorial(n) / factorial(n - r) * factorial(r);
}

int main() {
    int nEntered;
    int rEntered;
    std::cout << ">> To calculate the binomial coefficient , please enter the n: ";
    std::cin >> nEntered;
    std::cout << ">> please enter the r: ";
    std::cin >> rEntered;
    std::cout << "---> bin. coef. Of n:" << nEntered << ", r:" << rEntered << " ->"
              << calc_binomical_coefficient(nEntered, rEntered) << std::endl;
    return 0;
}
```

2.
Write a function **`permutNumbers`** that prints all n! many permutations of the numbers 1 to n on std::out.
Example: the output for **`permutNumbers(3)`** shall be: 123, 132, 213, 231, 312, 321

```
#include <iostream>

void swap(int &a, int &b) {
    int temp = a;
    a = b;
    b = temp;
}

void permute(int a[], int helper, int n) {

    if (helper == n) {
        for (int i = 0; i <= n; i++) {
            std::cout << a[i];
        }
        std::cout << " ";
    } else {
        // Permutations made
        for (int i = helper; i <= n; i++) {
            swap(a[helper], a[i]);
            permute(a, helper + 1, n);
            swap(a[helper], a[i]);
        }
    }
}

void permutateNumbers(int n) {
    int permutation[n];
    for (int i = 1; i <= n; i++) {
        permutation[i - 1] = i;
    }
    permute(permutation, 0, n - 1);
}

int main() {
    int nEntered;
    std::cout << ">> Please enter a number: ";
    std::cin >> nEntered;
    permutateNumbers(nEntered);
    return 0;
}
```

3. Given the following function definition:

```
int sum_down(int x)
{
     if (x >= 0)
     {
          x = x - 1;
          int y = x + sum_down(x);
          return y + sum_down(x);
     }
     else
     {
          return 1;
     }
}
```

a) What is this smallest integer value of the parameter **x**, so that the returned value is greater than 1.000.000 ?
The smallest integer is 19 (result of 19: 1 048 556, result of 18: 524 269).

b) Rewrite the function, so that it is free of recursion. I.e. give an iterative definition on the foundation of a loop-construct.
TIP: First transform the recursive definition, so that you have just a single recursive call.

```cpp
#include <iostream>

int sum_down_it(int x) {
    int temp = 1;
    int y;

    if (x <= 0)
        return temp;

    for (int i = 0; i < x; i++) {
        y = i + temp;
        y = y + temp;
        temp = y;
    }
    return y;
}

int main() {
    std::cout << "iterative " << sum_down_it(8) << "\n";
    return 0;
}
```

c) Is it OK to switch the type of the parameter **x** to **double**?
Discuss your decision / give an argumentation.
Yes, I think it is okay to switch the parameter to a double. Converting from one data type to another is a chance of data loss. But in our case, this does not occur, since

smaller-type data is transformed in larger-type data. A double is usually large enough to represent exactly any int.

A double is typically saved in 8 Bytes while an Integer is saved in 4. I think in most cases this wouldn't matter too much though, so I think its still fine too switch.

So, when changing the data type, you can still enter every integer that you could have entered before. Now you can even enter decimal numbers. You need to be aware of that though! The user can now enter decimal numbers and will get a result for them. Do we want this? Is the result that our program then computes the result that we want?

d) Is it OK to switch the type of the parameter **`x`** to **`unsigned int`**?
Discuss your decision / give an argumentation.
The answer to this question really depends on what your goal is. In our program now it is possible to enter a negative Integer and get a result for each of these numbers. If we switch to an unsigned int, negative numbers can't be entered anymore. If that is no problem for your goal, then you can change it, otherwise you need to stick to the signed data types.

Also, a signed integer can store a range of values
from **2,147,483,648** to **2,147,483,647**. Whereas an unsigned int variable can store a range of values from **0** to **4,294,967,295**. So not only do you lose the negative range if you switch to an unsigned integer, but you also gain a lot more possible positive range. Since these big numbers probably don't bring any disadvantages with it, that is probably fine though.

e) Is it OK to switch the function head to **`int sum_down(const int x)`**?
Discuss your decision / give an argumentation.
No, this is not okay. We are changing variable x during our program. This is not possible with a constant, as the name suggests.