Programming 101

1. Write a program that prints ”Hello World” and a newline character to stdout. Use the object `std::cout` from the header `iostream`.
2. Use local variables of types `std::string` and `int` to ask the user for their name and age. Now that you know this data, greet the user back. (You need to include the header `string`)
3. Ask the user for 2 integers - width and height of a rectangle. Use `std::cout` to print the area in a specific format. All numbers should align on the right. Fill with spaces if needed. Hint: Check out the documentation of `iomanip`.
4. Ask the user for 2 positive integers a and b. Print $a + b$, $a \cdot b$, $a \bmod b$. Now print a/b. Use `static_cast<T>()` to cast either a or b to a double and print the real division of a/b, too.

Functions

1. Implement the following functions in C++ and call them from main.

$$\text{sum} : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z} : (x, y) \mapsto x + y$$

$$\text{product} : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z} : (x, y) \mapsto x \cdot y$$

2. Write a function that takes an ASCII code and returns the corresponding character. Implement the inverse function, too. (Σ is the ASCII alphabet)

$$f : \mathbb{Z} \to \Sigma : x \mapsto c$$

$$f^{-1} : \Sigma \to \mathbb{Z} : c \mapsto x$$

3. Implement the following functions

$$f : \mathbb{N} \to \mathbb{N} : x \mapsto \sum_{n=1}^{x} n$$

$$g : \mathbb{N} \times \mathbb{N} \to \mathbb{N} : (x, y) \mapsto \sum_{n=x}^{y} n$$

$$h : \mathbb{N} \to \mathbb{N} : k \mapsto \prod_{i=1}^{k} i$$

$$p : \mathbb{N} \times \mathbb{N} \to \mathbb{N} : (n, k) \mapsto \binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$$

4. Write a function ”isPrime” that takes an integer n and returns a bool. If n is prime, return true. Return false otherwise.

Boolean operations

1. Implement a function `bool neg(bool x)` that returns x negated.
2. Write a function `bool f(int x, int y)` that returns true if and only if both x and y are not equal to 0.
3. Write a function `bool g(int x, int y)` that returns true if x is greater than y.
4. Write a function `bool h(int x)` that returns true if x is divisible by 2
5. Implement a logical XOR function `bool xor(bool a, bool b)` that returns $a \oplus b$ by using AND, OR, and NOT operations.
6. The function `bool nand(const bool* vals, int n)` takes an array of n boolean values and returns the collective NAND of all those values. Implement this function in C++.

Bitwise operations

1. You are writing a wrapper for the ”open” syscall which is used to open files. For now you want to simulate the behavior and try to evaluate the given flags. Your function prototype is `void open(std::string path, char flags)`. There are 4 possible flags (written in binary).

   - read: $(00000001)_2$
   - write: $(00000010)_2$
   - append: $(00000100)_2$
   - truncate:$(00001000)_2$

   The ”flags” argument can be any combination of those. Your simulated function should print the given path and which flags have been set. How can the user combine the flags?

2. Write a function `int ones(int n)` that returns the number of bits in n which are 1s.

3. Implement a function `bool f(int n)` that returns true if the 1st, 2nd, and 4th bit of n are set. (Hint: 1st bit is LSB)

4. Write a function `int power2(int n)` that returns the n-th power of 2 without using a loop inside the function.

Values and references

Create a custom namespace called `exc`.

1. Implement a function `exc::inc` that takes an integer and increments the original variable.

2. Write a function `exc::swap` that takes 2 integer variables and swaps their content. Make sure the function deals correctly with invalid arguments.

3. Implement a function `void exc::ror(int* arr, int len, int n)`, that rotates an integer array of ”len” elements by ”n” to the right. Your function should deal with invalid arguments for ”arr”, ”len” and ”n”.

Pointer arithmetics

The matrix $A \in \mathbb{Z}^{m \times n}$ is implemented as `int A[m][n]`. In memory this is of course equivalent to `int A[m * n]`. The following functions are defined inside the namespace `mat`.

1. Implement a function `mat::printFromRows(const int* A, int rows, int cols, int k)` that prints the k-th value of every row in matrix A. Handle invalid arguments accordingly.

2. Write a function `mat::printFromCols(const int* A, int rows, int cols, int k)` that prints the k-th value of every column in matrix A. Handle invalid arguments accordingly.

3. Write a function `mat::add(const int* A, const int* B, int* C, int rows, int cols` that takes 2 matrices A and B and puts their sum into matrix C. Handle invalid arguments, too.

Function overloading

Create the custom namespace `ovl`.

1. Implement the function `int ovl::max(int x, int y)` that returns x if x $>$ y or y otherwise. Overload this function for the datatype ”double”. Think about the return type, too. Each implementation of ”max” should print if it has been called with int or double arguments.

Memory manipulation (advanced)

1. An integer i consists of 4 bytes. Each byte can be manipulated seperately. Choose a datatype T with `sizeof(T) == 1`. Create a pointer of type T, set it to the address of i. Fix type mismatches by using C++ casts like `static_cast<T*>()`. Use pointer arithmetics to increment each of the 4 bytes by 1. Print the result.

2. Print the hex value of each byte of an integer to stdout. Interpret the result with the endianness of your system.

3. Use a char pointer to modify the exponent of a float variable. Take the endianness of your system into consideration.