// --------- TASK 1 ---------

// function which will return given amount of bits with desired shift

uint32 GetBitsFromUint32(unit32 inputValue, uint8 bitOffset, uint8 bitLength);

//example:

uint32 res = GetBitsFromUint32(0xABCDEF12, 4, 12);

// result is: res = 0x00000EF1;

// --------- TASK2 ---------

//overload function with const char \*:

uint32 GetBitsFromUint32(const char\* inputValue, uint8 bitOffset, uint8 bitLength);

uint32 res = GetBitsFromStr("0xABCDEF12", 4, 12); // pay attention to quotation!

// result shall be the same

// res = 0x00000EF1;

// --------- TASK 3 ---------

// Extract bool vector from overloaded function

std::vector<bool> GetBitsVector(std::string inputValue);

std::vector<bool> GetBitsVector(uint32 inputValue);

// usage:

std::vector<bool> res = GetBitsVector(149);

// or

std::vector<bool> res = GetBitsVector(0x95);

// or

std::vector<bool> res = GetBitsVector("0x95");

// expected result (index 0 = LSB)

// res = {true, false, true, false, true, false, false, true, ...}

// --------- TASK 4 ---------

// Create Tree class which will be able to generate string trees (like folders structure)

// example of usage:

int main(...)

{

Tree\* root = new Tree("tree name");

Tree\* galaz1 = root->AddSub("galaz 1"); // this function creates a child Tree object and returns pointer to it

Tree\* galaz2 = root->AddSub("galaz 2");

Tree\* galaz3 = root->AddSub("galaz 3");

Tree\* galaz1\_1 = galaz1->AddSub("galaz 1.1");

Tree\* galaz2\_1 = galaz2->AddSub("galaz 2.1");

Tree\* galaz2\_2 = galaz2->AddSub("galaz 2.2");

Tree\* galaz2\_1\_2 = galaz2\_1->AddSub("galaz 2.1.2");

// after execution

root->print(0); // in parameter pass initial indent for the object

// Expected result:

// tree name

// galaz 1

// galaz 1.1

// galaz 2

// galaz 2.1

// galaz 2.1.1

// galaz 2.2

// galaz 3

// print works recursive, where indent is passed

uint32 rootChildrenCnt = root->GetSubCount(); // result shall be 3

uint32 galaz1childrenCount = galaz1->GetSubCount(); // result shall be 1

uint32 countOfAllChildren = root->GetAllSubCount(); // result shall be 7 (recursive children counting)

root->Del(1); // will remove galaz\_2 with all it's children (recursive), so in a result only galaz\_1 and galaz\_3 will stay

uint32 rootChildrenCnt \_v2 = root->GetSubCount(); // result shall be 2

uint32 countOfAllChildren \_v2 = root->GetAllSubCount(); // result shall be 3 (recursive children counting)

delete(root); // deleting root shall remove all children (recursive) and clean memory

}

// --------- TASK 5 ---------

// Extension of "print" function which will add automatic numbering

// execution:

root->print(0, true); // deepth= 0, true = print numbers

// We shall get following result (root has no number)

// tree name

// 1. galaz 1

// 1.1. galaz 1.1

// 2. galaz 2

// 2.1. galaz 2.1

// 2.1.1. galaz 2.1.1

// 2.2. galaz 2.2

// 3. galaz 3

// remove second child:

root->Del(1);

// result:

// tree name

// 1. galaz 1

// 1.1. galaz 1.1

// 2. galaz 3

// ---------------- TASK 6 ---------------

// Develop function, which will show all elements of vector in reverse order without (!) usage of any loops or libraries.

// input vector: { a,b,c,d,e,f,g,h,i,j }

// output:

// j,i,h,g,f,e,d,c,b,a

// ---------------- TASK 7 ----------------

Prepare class / set of functions, which allows serialization and deserialization of binary data, like :

BitStream->Add(4, &Var0) // adds 4 bits of Var0 to a data stream / buffer

BitStream->Add(2, &Var1)// adds 2 bits of Var1 to a data stream / buffer

BitStream->Add(5, &Var2)// adds 5 bits of Var2 to a data stream / buffer

BitStream->Add(1, &Var3)// adds 1 bit of Var3 to a data stream / buffer

BitStream->Add(8, &Var4)// adds 8 bits of Var4 to a data stream / buffer

BitStream->Add(16, &Var5)// adds 16 bits of Var5 to a data stream / buffer

BitStream->Add(4, &Var6)// adds 4 bits of Var6 to a data stream / buffer

// Var0..6 can be any type

Function samples :

uint32 BitStream::Add(uint32 bitLength, void\* dataAddr); //returns actual position after adding data to stream

uint32 BitStream::Get(uint32 bitLength, void\* dataAddr); // returns position after reading bits and storing unter dataAddr

uint32 BitStream::GetBitLength(); // returns amount of added bits

uint32 BitStream::GetData(void\* addr, uint32 maxBitLength); // allows to get full stream, returns number of bits

BitStream::ResetData(); // clears internal buffer

