```
#include <string>
#include <cstring>
#include <cstdint>
#include <algorithm>
#include <type_traits>

template<typename T> struct ShortString
{
    static_assert(std::is_integral<T>:: value && std::is_unsigned<T>:: value,
    "StringAsUInt implemented only for integral unsigned types "  );
    enum {MaxSize = sizeof(T) -1 };
    static constexpr T Invalid() { return T() ; }
    ////////////
    ShortString(T value) noexcept  : m_value{value}  {}
    ShortString(const char* value, std::size_t len) noexcept{
        if(len <= MaxSize){  memcpy(&m_value, value, len); }
    }
    ShortString(const char* c_str) noexcept : ShortString(c_str, strlen(c_str) ){  }
    ShortString(const std::string &str) noexcept : ShortString(str.c_str(), str.length()
    )    { }
    ////////////
    bool IsValid() const noexcept{
        return (m_value != Invalid() );
    }
    ///////////////
    operator T() const noexcept {
        return m_value;
    }
    operator const char* ()  const noexcept {
        return reinterpret_cast<const char *> (&m_value) ;
    }
    operator std::string  ()  const noexcept {
        return reinterpret_cast<const char *> (&m_value) ;
    }
private:
    T  m_value{ Invalid() };
};

int main(){
    ShortString<std:: uint64_t> value1("SPY", 3);
    ShortString<std:: uint64_t> value2("SPY");
    ShortString<std:: uint64_t> value3(std::string("SPY"));
    std::string v11 = value1;   const char* v12 = value1;  uint64_t v13 = value1;
    std::string v21 = value2;
    std::string v31 = value3;

    ShortString<std:: uint64_t> value4("1234567");
    std::string v4 = value4;
    ShortString<std:: uint64_t> value5("12345678");
    std::string v5 = value5;
    return 0;
};
```