1 dailyLog

- type punning using union, reinterpret case, strict alialing rule
- union type punning, size of constexpr int == 4B or less than 4B

```
std::bitset<5> intA;
struct s1 {
    int a:6;
}
#include <iostream>
#include <stdexcept>
#include <functional>
#include <bitset>
#include <locale>
#include <stdio.h>
#include <codecvt>
#include <assert.h>
using namespace std::string_literals;
auto s1 = "abc";
auto s1 = L"abc";
auto s1 = u8"abc";
auto s1 = u"abc";
auto s1 = U"abc";
auto s1 = "abc";
auto s1 = "abc";
auto s1 = 42;
auto s1 = 42l;
auto s1 = 42ul;
auto s1 = 42ull;
auto s1 = 42.f;
auto s1 = 42.;
auto s1 = 42;
auto s1 = 0b11
auto s1 = 0123;
auto s1 = 0x123;
std::cout << std::hex << std::dec << std::oct << std::boolalpha
std::cout << std::bitset<5>(19) << std::endl;
std::setlocale(LC_ALL, "en_US.UTF-8");
std::u32string p1 = U"abc"s;
std::string p2 = "abc"s;
std::u16string p3 = u"abc"s;
enum e1{R,RED,YELLOW};
enum class e1{R,RED,YELLOW};
#define RED1 1
enum class e1:uint8_t
{
```

```
    R=253,
    RED,
    YELLOW
};

template <typename T> void type_name(){
    std::cout << __PRETTY_FUNCTION__ << std::endl;
}
string toUTF8(const basic_string<T, char_traits<T>, allocator<T>>& source)
{
    string result;
    wstring_convert<codecvt_utf8_utf16<T>, T> convertor;
    result = convertor.to_bytes(source);
    return result;
};

template <typename T>
auto fromUTF8(string source)
{
    wstring_convert<codecvt_utf8_utf16<T>, T> convertor;
    basic_string<T, char_traits<T>, allocator<T>>  result = convertor.from_bytes(source);
    return result;
}

//UB, no exception
 int x = 20;
 auto p = reinterpret_cast<float*>(&x);
```