```cpp

#include <memory>
#include <stdlib.h>
#include <type_traits>
#include <vector>
#include <iostream>

constexpr bool isPowerOf2(size_t n) {
    return ((n != 0) && (!(n & (n - 1))));
}

template <typename T> struct Deleter {
    void operator()(T* ptr) {
        if (!ptr) {
            return;
        }
        ptr -> ~T();
        free(ptr);
    }
};
template <typename T> using PTR = std::unique_ptr<T, Deleter<T> >;

template <typename T, size_t alignment, typename ...ARGS> inline PTR<T>
  make_aligned_unique(ARGS&& ... args) {
    static_assert(isPowerOf2(alignment), "Alignment should be power of 2");
    static_assert(alignment % (sizeof(void*)) == 0, "Alignment should be
      multiple of sizeof(void *)");

    void* buf = nullptr;
    posix_memalign(&buf, alignment, sizeof(T));
    if (!buf) {
        return PTR<T>();
    }

    PTR<T> ptr(new(buf) T(std::forward<ARGS>(args) ...));
    return ptr;
}


int main() {

    auto p1 = make_aligned_unique<int, 16>(3);
    auto p2 = make_aligned_unique<int, 16>(5);
    auto p3 = make_aligned_unique<int, 32>(9);

    std::vector<PTR <int> > vec;
    vec.push_back(std::move(p1)); vec.push_back(std::move(p2)); vec.push_back
      (std::move(p3));
    for (auto & p : vec) {
```

```
47          std::cout << *p << std::endl;
48      }
49
50
51  }
```