1) **Короткий опис проблеми («назва»), за якою можна більш-менш зрозуміти, про що іде мова**

«Як за допомогою недоцільної назви змінної та макросу можна забезпечити собі дебагінг на цілу годину(години)»

2) **Детальний опис, в чому саме полягає проблема чи незвична ситуація**

Нехай у нас є простий unit tests framework test_runner.h:

```
//test_runner.h

#pragma once

#include <sstream>
#include <stdexcept>
#include <iostream>
#include <map>
#include <set>
#include <string>
#include <vector>

using namespace std;

//тут були перевизначені оператори для виводу в консоль певних контейнерів, але я
//їх прибрав для зручності

template<class T, class U>
void AssertEqual(const T& t, const U& u, const string& hint = {}) {
  if (!(t == u)) {
    ostringstream os;
    os << "Assertion failed: " << t << " != " << u;
    if (!hint.empty()) {
       os << " hint: " << hint;
    }
    throw runtime_error(os.str());
  }
}

inline void Assert(bool b, const string& hint) {
  AssertEqual(b, true, hint);
}

class TestRunner {
public:
  template <class TestFunc>
  void RunTest(TestFunc func, const string& test_name) {
    try {
      func();
      cerr << test_name << " OK" << endl;
    } catch (exception& e) {
```

```
      ++fail_count;
      cerr << test_name << " fail: " << e.what() << endl;
    } catch (...) {
      ++fail_count;
      cerr << "Unknown exception caught" << endl;
    }
  }

  ~TestRunner() {
    if (fail_count > 0) {
      cerr << fail_count << " unit tests failed. Terminate" << endl;
      exit(1);
    }
  }

private:
  int fail_count = 0;
};

#define ASSERT_EQUAL(x, y) {            \
  ostringstream os;                     \
  os << #x << " != " << #y << ", "      \
    << __FILE__ << ":" << __LINE__;     \
  AssertEqual(x, y, os.str());          \
}

#define ASSERT(x) {                     \
  ostringstream os;                     \
  os << #x << " is false, "             \
    << __FILE__ << ":" << __LINE__;     \
  Assert(x, os.str());                  \
}

#define RUN_TEST(tr, func) \
  tr.RunTest(func, #func)
```

Він містить дві функції : `AssertEqual`, та `Assert,` які перевіряють на рівність дві змінні; клас `TestRunner,`що запускає тести і при паданні тесту виводить інформацію про його, та зупиняє роботу програми, якщо хоча б один тест впав.
Також, test_runner.h містить три макроси, які автоматизують формування підказок при паданні тесту.

Тепер розглянемо наступний блок коду

```
void TestLooping() {
  vector<int> v(15);
```

```
  iota(begin(v), end(v), 1);

 //Paginator – шаблон класу, який бере діапазон контейнера та «нарізає»
 //на «сторінки» - діапазон фіксованого розміру.
 //Його реалізацію можна глянути тут :
 // https://github.com/ram333n/cppcoursera/blob/main/Red/Week%201/paginator.cpp

  Paginator<vector<int>::iterator> paginate_v(v.begin(), v.end(), 6);
  ostringstream os;
  for (const auto& page : paginate_v) {
    for (int x : page) {
      os << x << ' ';
    }
    os << '\n';
  }

  ASSERT_EQUAL(os.str(), "1 2 3 4 5 6 \n7 8 9 10 11 12 \n13 14 15 \n");
}
```

Це unit test, який перевіряє можливість ітерування циклом range based for по об'єкту класу Paginator. Якщо взяти правильну реалізацію шаблона класу Paginator і запустити цей unit test, то він впаде і програма завершить роботу не з нульовим кодом виходу.

Здавалося, правильна реалізація шаблона класу падає на цьому тесті. Але, давайте розберемося детальніше.
Як бачимо, в цьому тесті ми особливо дивного нічого не робимо, окрім однієї штуки : перевірки на рівність через макрос `ASSERT_EQUAL`. Запишемо, в що цей макрос розкривається :

```
void TestLooping() {
  vector<int> v(15);
  iota(begin(v), end(v), 1);

  Paginator<vector<int>::iterator> paginate_v(v.begin(), v.end(), 6);
  ostringstream os;
  for (const auto& page : paginate_v) {
    for (int x : page) {
      os << x << ' ';
    }
    os << '\n';
  }
  {
    ostringstream os;
    os << os.str() << " != " << "1 2 3 4 5 6 \n7 8 9 10 11 12 \n13 14 15 \n" << ","
       << /*назва файлу*/ << ":" << /*номер стрічки, у якій викликається макрос*/ ;
    AssertEqual(os.str(), "1 2 3 4 5 6 \n7 8 9 10 11 12 \n13 14 15 \n", os.str()) ;
  }
}
```

Замість того, щоб порівняти результат роботи тесту у змінній os, яку ми створили перед вкладеними циклами, ми порівнюємо технічну інформацію з очікуваним результатом. Через це тест і падає.

3)Опис вирішення проблеми (якщо відомий).

Найпростіший спосіб – це просто змінити назви змінних у макросах на змістовні, які будуть рідко використовуватися в інших блоках коду, тобто os замінити, наприклад, на assertion_equal_hint_message.

4)Короткий приклад коду, що ілюструє дану проблему.

Цей код для того, щоб запустити і побачити проблему.

```
#include <sstream>
#include <stdexcept>
#include <iostream>
#include <map>
#include <set>
#include <string>
#include <vector>
#include <numeric>

using namespace std;

template<class T, class U>
void AssertEqual(const T& t, const U& u, const string& hint = {}) {
  if (!(t == u)) {
    ostringstream os;
    os << "Assertion failed: " << t << " != " << u;
    if (!hint.empty()) {
       os << " hint: " << hint;
    }
    throw runtime_error(os.str());
  }
}

inline void Assert(bool b, const string& hint) {
  AssertEqual(b, true, hint);
}

class TestRunner {
public:
  template <class TestFunc>
  void RunTest(TestFunc func, const string& test_name) {
    try {
      func();
      cerr << test_name << " OK" << endl;
    } catch (exception& e) {
      ++fail_count;
      cerr << test_name << " fail: " << e.what() << endl;
    } catch (...) {
      ++fail_count;
      cerr << "Unknown exception caught" << endl;
    }
  }

  ~TestRunner() {
    if (fail_count > 0) {
      cerr << fail_count << " unit tests failed. Terminate" << endl;
      exit(1);
    }
  }
```

```
private:
  int fail_count = 0;
};

#define ASSERT_EQUAL(x, y) {            \
  ostringstream os;                     \
  os << #x << " != " << #y << ", "      \
    << __FILE__ << ":" << __LINE__;     \
  AssertEqual(x, y, os.str());          \
}

#define ASSERT(x) {                     \
  ostringstream os;                     \
  os << #x << " is false, "             \
    << __FILE__ << ":" << __LINE__;     \
  Assert(x, os.str());                  \
}

#define RUN_TEST(tr, func) \
  tr.RunTest(func, #func)


template<typename Iterator>
struct IteratorRange {
public:
    IteratorRange(Iterator f, Iterator l)
        : first(f), last(l) {};

    Iterator begin() const {
        return first;
    }

    Iterator end() const {
        return last;
    }

    size_t size() const {
        return last - first;
    }

private:
    Iterator first, last;
};

template <typename Iterator>
class Paginator {
public:
    Paginator(Iterator begin, Iterator end, size_t page_size) {
        for (size_t range_size = end - begin; range_size;) {
            size_t step = min(page_size, range_size);
            pages.push_back({ begin,begin + step });
            range_size -= step;
            begin += step;
        }

    }

    auto begin() const {
        return pages.begin();
    }

    auto end() const {
        return pages.end();
    }

    size_t size() const {
        return pages.size();
    }

private:
    vector<IteratorRange<Iterator>> pages;
```

```
};

template <typename C>
auto Paginate(C& c, size_t page_size) {
    return Paginator(c.begin(), c.end(), page_size);
}


void TestLooping() {
    vector<int> v(15);
    iota(begin(v), end(v), 1);

    Paginator<vector<int>::iterator> paginate_v(v.begin(), v.end(), 6);
    ostringstream os;
    for (const auto& page : paginate_v) {
        for (int x : page) {
            os << x << ' ';
        }
        os << '\n';
    }

    ASSERT_EQUAL(os.str(), "1 2 3 4 5 6 \n7 8 9 10 11 12 \n13 14 15 \n");
}


int main() {
    TestRunner tr;
    RUN_TEST(tr, TestLooping);
}
```

5) **Посилання на репозиторії, де можна побачити цю проблему в реальному коді (якщо такі є)**

https://github.com/ram333n/cppcoursera/blob/main/Red/Week%201/paginator.cpp - функція TestLooping (77-90 lines)

7) **Ключові слова, за якими варто шукати інформацію про цю проблему.**

C macros pitfalls

8) **Чому ця проблема здалась Вам цікавою?**

На маю думку, ця проблема здалась цікавою через свою незвичну поведінку та відносну «неочевидність». Щоб її побачити, треба було «перерити» всі файли з кодом, запускати препроцесінг та аналізувати його роботу.