

Politechnika Krakowska
przedmiot: Systemy Odporne na Błędy
Laboratorium 1: Asercja w języku C/C++
sprawozdanie
wykonał: Tomasz Fudalej

1. Wstęp

Celem laboratorium jest stworzenie listy dwukierunkowej w języku C lub C++ i uodpornienie jej na błędy przy pomocy mechanizmu asercji, a następnie wykonanie jego testów.

- **Lista dwukierunkowa:**

Lista dwukierunkowa to struktura, której każdy element posiada odwołanie do następnego i poprzedniego elementu struktury.

- **Asercja:**

W językach C i C++ przybierają formę funkcji `assert(boolVar)`, gdzie `boolVar` to zmienna logiczna. Jeśli argument funkcji `assert` ma wartość `false`, następuje przerwanie działania programu i wyświetlenie komunikatu z informacją o miejscu w kodzie, który wywołał błąd.

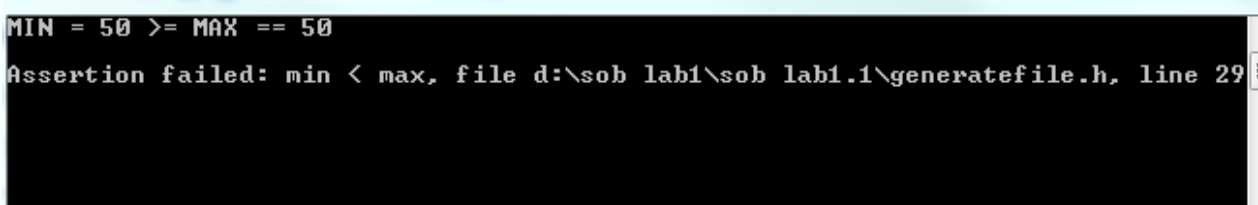
2. Przykłady asercji:

- generacja danych wejściowych
 - Parametry danych wejściowych:

```
template<typename Type>
GenerateFile<Type>::GenerateFile(std::string path, int min, int max, int count)
{
    /*if (min >= max)
        cout << "MIN = " << min << " >= MAX == " << max << endl << endl;*/
    assert(min < max);
    assert(count > 0);
    this->min = min;
    this->max = max;
    this->count = count;
}
```

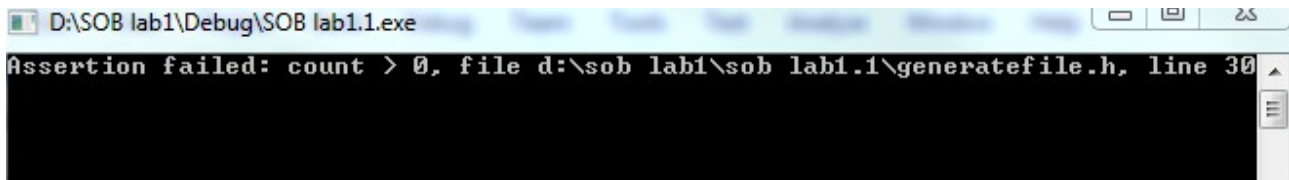
Przy pomocy asercji sprawdzana jest poprawność zakresu danych. W szczególności chroni to przed dzieleniem przez zero w trakcie generowania danych pseudolosowych.

Wynik testu:



```
MIN = 50 >= MAX == 50
Assertion failed: min < max, file d:\sob lab1\sob lab1.1\generatefile.h, line 29
```

Sprawdzana jest też poprawność ilości potrzebnych do wygenerowania danych wejściowych (`count`).



- generacja danych wejściowych

```
template<typename Type>
void GenerateFile<Type>::generateFile(string path)
{
    srand(time(NULL));
    std::ofstream file(path);
    assert(file.good());

    if (is_integral<Type>::value) {
        for (int i = 0; i < count; i++) {
            file << rand() % (max - min) + min << endl;
        }
    }
    else if (is_floating_point<Type>::value) {
        for (int i = 0; i < count; i++) {
            file << (static_cast<float>(rand()) / static_cast<float>(RAND_MAX)) * (max - min) + min << endl;
        }
    }
    else
        assert(false);

    file.close();
}
```

Pierwsze użycie asercji sprawdza poprawność otwarcia nowego pliku, w którym umieszczone zostaną wygenerowane dane.

Drugie użycie asercji zapewnia, że podany typ przechowywany w liście dwukierunkowej jest albo typem *int*, albo typem *float*.

- odczytywanie danych z pliku wejściowego:

```
template <typename Type>
void ReadFile<Type>::readFile(string path, qu::Queue<Type> &queue, const Type min, const Type max)
{
    std::fstream file;
    file.open(path, std::ios::in);
    assert(file.good());
    string s;

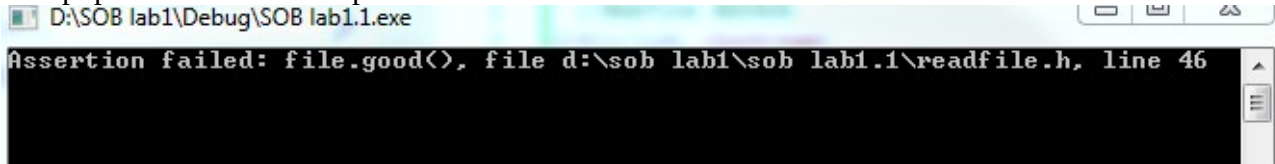
    while (true)
    {
        file >> s;
        if (!file.eof()) {
            auto d = stringToValue<Type>(s);
            QueueElement<Type> *newEl = new QueueElement<Type>(d);
            assert(newEl);
            assert(queue.addWithBoundary(newEl, min, max));
        }
        else
            break;
    }
}
```

Pierwsza asercja sprawdza poprawność otwarcia pliku wejściowego.

Druga asercja sprawdza poprawność utworzenia nowego elementu lista przy pomocy operatora *new*.

Trzecia asercja sprawdza poprawność dodania elementu do listy.

Test poprawności otwarcia pliki:



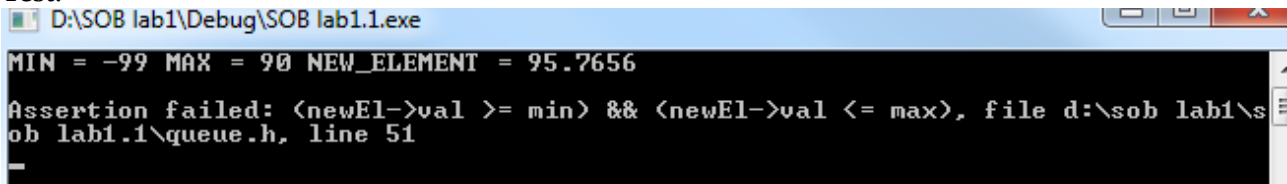
- lista dwukierunkowa

```
template<typename Value>
bool Queue<Value>::addWithBoundary(QueueElement<Value>* newEl, const int min, const int max)
{
    assert(std::is_integral<Value>::value || std::is_floating_point<Value>::value);
    /*if (!(newEl->val >= min) && (newEl->val <= max))
        cout << "MIN = " << min << " MAX = " << max << " NEW_ELEMENT = " << newEl->val << endl << endl;*/
    assert((newEl->val >= min) && (newEl->val <= max));
}
```

Podczas dodawania elementów do listy, asercją sprawdzane jest, czy typ przechowywanej przez element wartości jest typem *int* lub *float*.

Następnie sprawdzana jest zgodność, czy otrzymana wartość do przechowania mieści się w zakresie.

Test:



- Użycie makra ***#define NDEBUG*** pozwala na wyłączenie działania asercji w programie i jego dalsze działania, pomimo zaistnienia błędów. W poniższym przykładzie, pomimo przekroczenia zakresu danych wejściowych, program kontynuuje pracę.

