

**POLITECHNIKA KRAKOWSKA**  
WYDZIAŁ INŻYNIERII ELEKTRYCZNEJ I KOMPUTEROWEJ



# **Asercja w języku C/C++**

Systemy odporne na błędy  
Sprawozdanie z laboratorium nr 1

Maria Guz  
Karol Pątko  
Wojciech Maludziński

# Wstęp

Asercja jest prostą formą testu akceptacyjnego. Decydując się na korzystanie z asercji zakładamy, że dany predykat powinien być prawdziwy. Sytuacja przeciwna powoduje przerwanie działania programu. Jedną z zalet używania mechanizmu asercji jest możliwość zidentyfikowania, gdzie dokładnie nastąpił błąd, dlatego dobrze sprawdza się przy testowaniu kodu.

## Cel zajęć

Celem zajęć laboratoryjnych nr 1 jest zapoznanie studentów z mechanizmem asercji oraz jego implementacja w języku C lub C++. Poza opracowaniem teoretycznym tematu studenci mają obowiązek przygotować praktyczne zadanie ukazujące działanie asercji w języku C/C++.

## Treść zadania

*Przygotować program w języku C lub C++, który odczyta dane liczbowe z podanego pliku (w formacie tekstowym lub binarnym) i umieści je w uporządkowanej liście dwukierunkowej. Format danych: [liczby całkowite, zmiennoprzecinkowe], zakres do własnego wyboru.*

*Funkcje programu:*

- *plik z danymi i zakres liczb podajemy, jako argumenty wywołania programu,*
- *dodawanie odczytanych liczb z pliku do dynamicznej listy dwukierunkowej uporządkowanej w kolejności rosnącej,*
- *dodawanie do listy tylko liczb z podanego zakresu,*
- *wyświetlanie zawartości listy od lewej do prawej i od prawej do lewej,*
- *usuwanie listy (przed zakończeniem programu).*

*Program powinien być wyposażony w mechanizmy odporności na błędy, m.in.:*

- *sprawdzanie efektów otwarcia (lub nie) wskazanego pliku;*
- *sprawdzanie zakresów liczb odczytanych z pliku z pomocą asercji;*
- *sprawdzanie wskaźników zwracanych przez funkcje przydzielające pamięć (malloc, new) z pomocą asercji;*

*Sprawdzić działanie gotowego programu w sposób symulujący błędy (brak pliku, niepoprawne dane) z włączoną oraz wyłączoną asercją (symbol NDEBUG).*

## Rozwiązanie zadania

Kod został rozdzielony na pliki *Element.h*, *Element.cpp*, *List.h*, *List.cpp* oraz *main.cpp*. W plikach *.h* znajdują się deklaracje struktur. W plikach *Element.cpp* i *List.cpp* umieszczone zostały definicje metod.

### Element.h

```
#ifndef Element_h
#define Element_h

#include <iostream>
#include <string>
#include <cstdlib>
#include <fstream>

// #define NDEBUG
#include <cassert>

using namespace std;

/* Struktura przechowująca pojedynczy obiekt listy */
struct el {
    double v;
    el* next;
    el* prev;
    el();
    el(double v);
    virtual ~el();
};

#endif
```

### Element.cpp

```
#include "Element.h"
/* Konstruktor elementu bez parametru */
el::el() {
    next = NULL;
    prev = NULL;
}

/* Konstruktor elementu z parametrem */
el::el(double temp) {
    assert(temp != NULL);
    next = NULL;
    prev = NULL;
    this->v = temp;
}

/* Destruktor elementu */
el::~el() {
    el* a = this->prev;
    el* b = this->next;
    if (a != NULL)
        a->next = b;
    if (b != NULL)
        b->prev = a;
}
```

## List.h

```
#ifndef List_h
#define List_h

#include "Element.h"
/*Klasa przechowująca listę*/
class List {
    el* head;
    el* tail;

public:
    List();
    virtual ~List();
    el* add(double a);
    void showRightToLeft();
    void showLeftToRight();
};

#endif
```

## List.cpp

```
#include "List.h"

/* konstruktor listy bez parametru */
List::List() {
    head = NULL;
    tail = NULL;
}

/* Destruktor listy*/
List::~List() {
    el* e = this->head;

    while (e != NULL) {
        el* temp = e;
        e = e->next;
        assert(temp != NULL);
        delete temp;
    }
}

/* Funkcja dodająca element do listy, zwraca wskaźnik do dodanego elementu. */
/// \param _a
/// \return
el* List::add(double _a) {
    el* current = new el(_a);
    assert(current != NULL);

    if (this->head == NULL) {
        this->head = current;
        this->tail = current;
        return current;
    }

    if (current->v < (this->head->v) {
        el* e = this->head;
        current->next = e;
        e->prev = current;
        this->head = current;
        return current;
    }

    el* next = this->head->next;
    while (next != NULL && current->v > next->v) {
        next = next->next;
    }
    if (next == NULL) {
        el* e = this->tail;
        e->next = current;
        current->prev = e;
        this->tail = current;
        return current;
    }

    el* a = next->prev;
    el* b = next;
    a->next = current;
```

```

        b->prev = current;
        current->prev = a;
        current->next = b;
        return current;
    }

    /* Funkcja wyświetlająca listę od lewej do prawej */
    void List::showLeftToRight() {
        el* e = this->head;
        while (e != NULL) {
            cout << e->v << " ";
            e = e->next;
        }
        cout << endl;
    }

    /* Funkcja wyświetlająca listę od prawej do lewej */
    void List::showRightToLeft() {
        el* e = this->tail;
        while (e != NULL) {
            cout << e->v << " ";
            e = e->prev;
        }
        cout << endl;
    }
}

```

## main.cpp

```

#include "Element.cpp"
#include "List.cpp"
#include <iostream>
#include <string>
#include <cstdlib>
#include <fstream>

using namespace std;

/*Funkcja sprawdzająca, czy wartość argumentu val mieści się w podanym zakresie (min-max), jeśli tak, dodaje
element listy.*/
/// \param val
/// \param min
/// \param max
/// \param list
void checkRangeAndAddToList(double val, int min, int max, List* list) {
    assert(list != null);
    assert(max > min);
    assert(val >= min && val <= max);

    if(val >= min && val <= max) {
        list->add(val);
        cout << val << " is in range " << min << ":" << max << "." << endl;

        assert(list != null);
    }
}

/*Funkcja sprawdzająca czy wprowadzony argument jest liczbą, zwracający prawdę dla liczb i fałsz dla reszty */
/// \param i
/// \param str
/// \return
bool isNumber(double& i, const string& str) {
    try {
        size_t pos;
        i = stod(str, &pos);
        return pos == str.size();
    }
    catch (const std::invalid_argument&) {
        return false;
    }
}

int main(int argc, char* argv[]){
    assert(argc == 4);

    string fileName = argv[1];
    int min = atoi(argv[2]);
    int max = atoi(argv[3]);

    ifstream file;
    file.open(fileName, std::fstream::in);
}

```

```

    assert(file.good());

    List* list = new List;
    assert(list != NULL);

    string number;
    while (!file.eof()) {
        getline(file, number);
        double i;

        bool isCorrect = isNumber(i, number);
        assert(isCorrect);
        checkRangeAndAddToList(i, min, max, list);
    }

    file.close();

    cout << "===== " << endl;
    cout << "List from left to right: " << endl;
    list->showLeftToRight();

    cout << "===== " << endl;
    cout << "List from right to left: " << endl;
    list->showRightToLeft();

    delete list;
    system("PAUSE");
    return 0;
}

```

# Testowanie programu

Program przyjmuje następujące argumenty:

```
Program arguments: "test.txt" 0 10000000
```

Plik tekstowy *test.txt* jest plikiem zawierającym dane wprowadzane do programu. Poprawnymi danymi są liczby całkowite i zmiennoprzecinkowe. Program ma za zadanie odczytać dane z pliku *test.txt* i wpisać je do dynamicznej listy dwukierunkowej posortowane od najmniejszej wartości do największej. Po wykonaniu tych operacji zawartość listy zostanie wyświetlona. Argumenty *0* i *10000000* są minimalną i maksymalną wartością, która może zostać wpisana do listy.

## Przypadek 1

Podczas wykonywania programu plik tekstowy *test.txt* istnieje. We wspomnianym pliku znajdują się poprawne dane - liczby całkowite i zmiennoprzecinkowe. Wszystkie liczby mieszczą się w zakresie podanym jako argument przy uruchamianiu programu. Asercja jest włączona.

*test.txt*:

1	12
2	23
3	234
4	543
5	12
6	23456
7	876
8	43
9	2345
10	64
11	324.34
12	34.4
13	2353.90
14	34256
15	2
16	1

output:

```
C:\Users\Mania\CLionProjects\untitled\cmake-build-debug\untitled.exe test.txt 0 10000000
12 is in range 0:10000000.
23 is in range 0:10000000.
234 is in range 0:10000000.
543 is in range 0:10000000.
12 is in range 0:10000000.
23456 is in range 0:10000000.
876 is in range 0:10000000.
43 is in range 0:10000000.
2345 is in range 0:10000000.
64 is in range 0:10000000.
324.34 is in range 0:10000000.
34.4 is in range 0:10000000.
2353.9 is in range 0:10000000.
34256 is in range 0:10000000.
2 is in range 0:10000000.
1 is in range 0:10000000.
=====
List from left to right:
1 2 12 12 23 34.4 43 64 234 324.34 543 876 2345 2353.9 23456 34256
=====
List from right to left:
34256 23456 2353.9 2345 876 543 324.34 234 64 43 34.4 23 12 12 2 1
Press any key to continue . . .
```

Wszystkie wartości mieszczą się w podanym jako argument zakresie i mają dopuszczalny typ, dlatego każda z nich została wpisana do listy. Lista dwukierunkowa jest uporządkowana. Wyświetlono wartości listy od najmniejszej do największej oraz od największej do najmniejszej.

## Przypadek 2

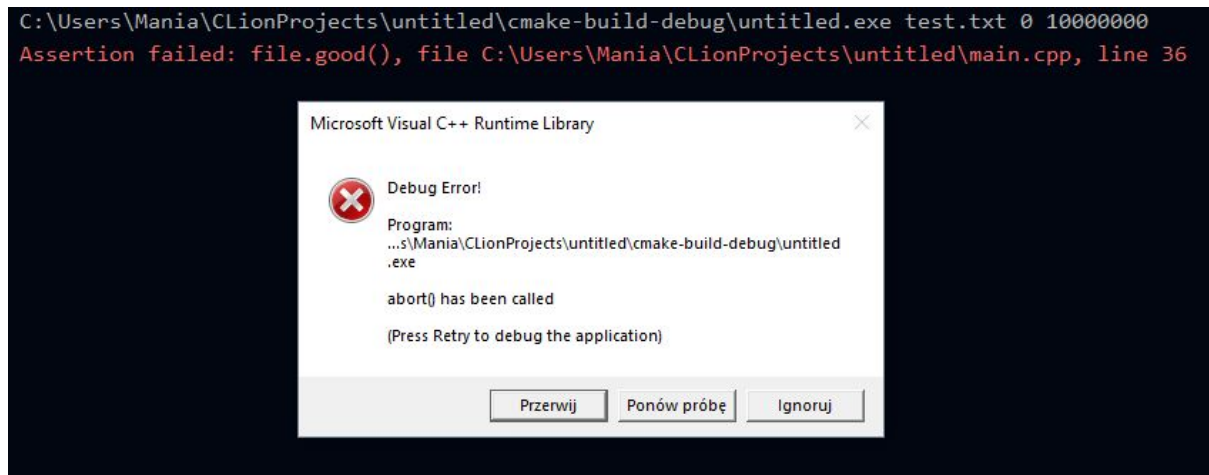
Podczas wykonywania programu plik tekstowy *test.txt* nie istnieje. Nazwa pliku przechowującego dane została zmieniona. Asercja jest włączona.

Fragment kodu odpowiedzialny za testowanie poprawności otwarcia pliku:

```
ifstream file;
file.open(fileName, std::fstream::in);
assert(file.good());
```



output:



W tym przypadku działa mechanizm asercji, który nie dopuszcza do dalszego działania programu, jeśli plik z danymi nie zostanie znaleziony.

### Przypadek 3

Podczas wykonywania programu plik tekstowy *test.txt* istnieje. We wspomnianym pliku znajdują się niepoprawne dane - liczby całkowite, zmiennoprzecinkowe, litery oraz inne znaki. Wszystkie liczby, poza błędnymi danymi, mieszczą się w zakresie podanym jako argument przy uruchamianiu programu. Asercja jest włączona.

Fragment kodu odpowiedzialny za testowanie poprawności danych:

```
string number;
while (!file.eof()) {
    getline(file, number);
    double i;

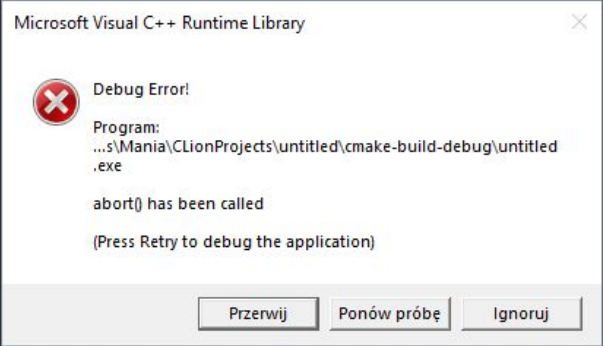
    bool isCorrect = isNumber(i, number);
    assert(isCorrect);
    checkRangeAndAddToList(i, min, max, list);
}
```

test.txt:

```
1      12
2      23
3      234
4      543
5      12
6      23456
7      876
8      43
9      2345
10     64
11     324.34
12     34.4
13     ~~~~~
14     ~~~~~
15     ~~~~~
16     2353.90
17     34256
18     2
19     1
```

output:

```
C:\Users\Mania\CLionProjects\untitled\cmake-build-debug\untitled.exe test.txt 0 10000000
12 is in range 0:10000000.
23 is in range 0:10000000.
234 is in range 0:10000000.
543 is in range 0:10000000.
12 is in range 0:10000000.
23456 is in range 0:10000000.
876 is in range 0:10000000.
43 is in range 0:10000000.
Assertion failed: isCorrect, file C:\Users\Mania\CLionProjects\untitled\main.cpp, line 47
2345 is in range 0:10000000.
64 is in range 0:10000000.
324.34 is in range 0:10000000.
34.4 is in range 0:10000000.
```



The error dialog box is titled "Microsoft Visual C++ Runtime Library". It features a red "X" icon and the text "Debug Error!". Below this, it specifies the program path: "...s\Mania\CLionProjects\untitled\cmake-build-debug\untitled.exe". The error message states "abort() has been called" and provides a hint: "(Press Retry to debug the application)". At the bottom, there are three buttons: "Przerwij" (Cancel), "Ponów próbę" (Retry), and "Ignoruj" (Ignore).

W tym przypadku działa mechanizm asercji, który nie dopuszcza do dalszego działania programu, jeśli plik zawiera niepoprawne dane.

## Przypadek 4

Podczas wykonywania programu plik tekstowy *test.txt* istnieje. We wspomnianym pliku znajdują się poprawne dane - liczby całkowite i zmiennoprzecinkowe. Niektóre z nich nie mieszczą się w zakresie podanym jako argument przy uruchamianiu programu. Asercja jest włączona.

Fragment kodu odpowiedzialny za sprawdzenie czy dana wartość mieści się w zakresie:

```
void checkRangeAndAddToList(double val, int min, int max, List* list) {
    assert(list != null);
    assert(max > min);
    assert(val >= min && val <= max);

    if(val >= min && val <= max) {
        list->add(val);
        cout << val << " is in range " << min << ":" << max << "." << endl;

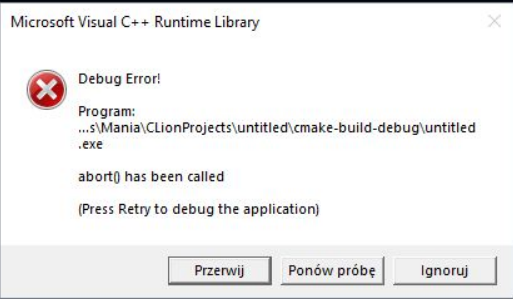
        assert(list != null);
    }
}
```

*test.txt*

1	12
2	23
3	234
4	543
5	12
6	23456
7	876
8	43
9	2345
10	64
11	324.34
12	34.4
13	23530000000000000000.90
14	34256
15	2
16	1

output:

```
C:\Users\Mania\CLionProjects\untitled\cmake-build-debug\untitled.exe test.txt 0 10000000
Assertion failed: val >= min && val <= max, file C:\Users\Mania\CLionProjects\untitled\main.cpp, line 18
12 is in range 0:10000000.
23 is in range 0:10000000.
234 is in range 0:10000000.
543 is in range 0:10000000.
12 is in range 0:10000000.
23456 is in range 0:10000000.
876 is in range 0:10000000.
43 is in range 0:10000000.
2345 is in range 0:10000000.
64 is in range 0:10000000.
324.34 is in range 0:10000000.
34.4 is in range 0:10000000.
```



W tym przypadku została sprawdzona poprawność parametrów podanych przy uruchamianiu programu oraz zakres liczb odczytanych z pliku. Mechanizm asercji nie dopuszcza do dalszego działania programu, jeśli liczba nie mieści się w zakresie.

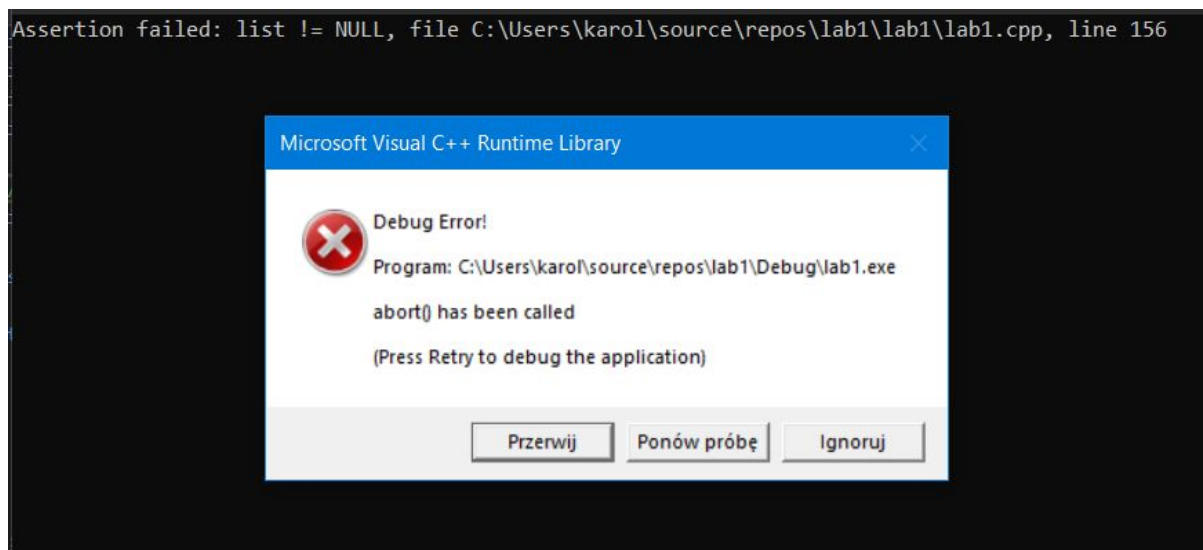
## Przypadek 5

Podczas wykonywania programu plik tekstowy *test.txt* istnieje. We wspomnianym pliku znajdują się poprawne dane - liczby całkowite i zmiennoprzecinkowe. Wszystkie liczby mieszczą się w zakresie podanym jako argument przy uruchamianiu programu. Dla zmiennej *list* nie została poprawnie przydzielona pamięć. Asercja jest włączona.

Fragment kodu odpowiedzialny za sprawdzanie wskaźników zwracanych przez funkcję przydzielającą pamięć.

```
List* list = new List;
assert(list != NULL);
```

output:



W tym przypadku działa mechanizm asercji, który nie dopuszcza do dalszego działania programu, jeśli lista została niepoprawnie utworzona lub zmodyfikowana.

## Przypadek 6

Podczas wykonywania programu plik tekstowy *test.txt* istnieje. We wspomnianym pliku znajdują się poprawne dane - liczby całkowite i zmiennoprzecinkowe. Niektóre z nich nie mieszczą się w zakresie podanym jako argument przy uruchamianiu programu. Asercja w tym przypadku jest jednak wyłączona.

```
#define NDEBUG
```

*test.txt*:

```
12
23
234
543
12
23456
876
43
2345
64
324.34
34.4
235300000000000000.9d
34256
2
1
```

output:

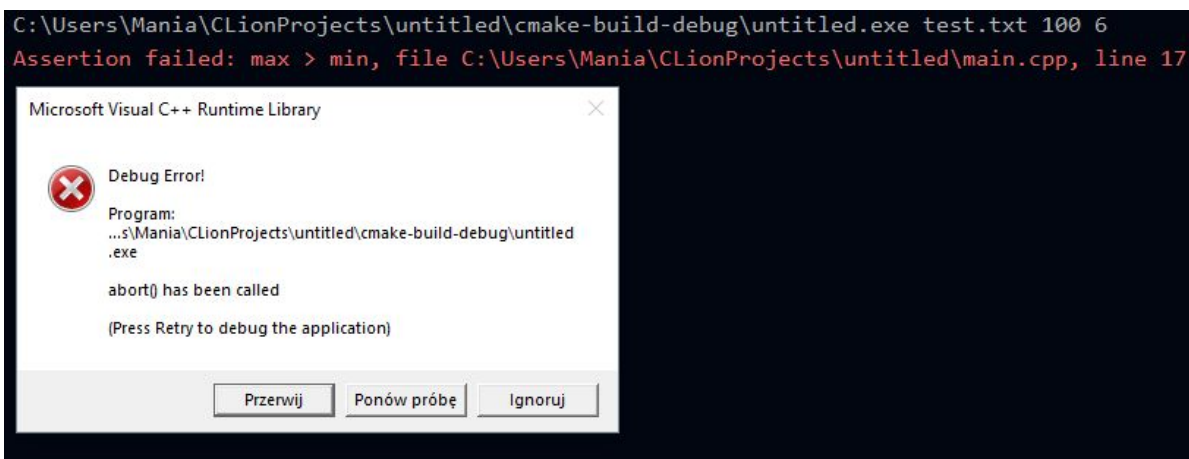
```
C:\Users\Mania\CLionProjects\untitled\cmake-build-debug\untitled.exe test.txt 0 10000000
12 is in range 0:10000000.
23 is in range 0:10000000.
234 is in range 0:10000000.
543 is in range 0:10000000.
12 is in range 0:10000000.
23456 is in range 0:10000000.
876 is in range 0:10000000.
43 is in range 0:10000000.
2345 is in range 0:10000000.
64 is in range 0:10000000.
324.34 is in range 0:10000000.
34.4 is in range 0:10000000.
34256 is in range 0:10000000.
2 is in range 0:10000000.
1 is in range 0:10000000.
=====
List from left to right:
1 2 12 12 23 34.4 43 64 234 324.34 543 876 2345 23456 34256
=====
List from right to left:
34256 23456 2345 876 543 324.34 234 64 43 34.4 23 12 12 2 1
Press any key to continue . . .
```

## Przypadek 7

Podczas wykonywania programu plik tekstowy *test.txt* istnieje. We wspomnianym pliku znajdują się poprawne dane - liczby całkowite i zmiennoprzecinkowe. Jednak zostały zmienione parametry podawane przy uruchamianiu programu tak, żeby wartość minimalna była większa niż wartość maksymalna. Asercja jest włączona.

Program arguments: "test.txt" 100 6

output:



## Wnioski

Asercja jest nieocenionym narzędziem w sprawdzaniu czy program nie zawiera błędów. Znacznie różni się od mechanizmu wyjątków, który to pozwala na utworzenie fragmentu kodu odpowiedzialnego za obsługę sytuacji wyjątkowej, a niekoniecznie błędu. Asercja nie daje możliwości implementacji obsługi zdarzenia w miejscu wystąpienia błędu. Jest jedynie informacją o zaistnieniu określonego błędu wraz z wskazaniem na miejsce jego wystąpienia co pozwala na szybkie zidentyfikowanie przyczyny. Dodatkowo mechanizm ten może zostać po prostu wyłączony w momencie ukończenia tworzenia oprogramowania bez konieczności usuwania wszystkich linii kodu odpowiedzialnych za przeprowadzenie testów co umożliwia szybsze zakończenie prac.