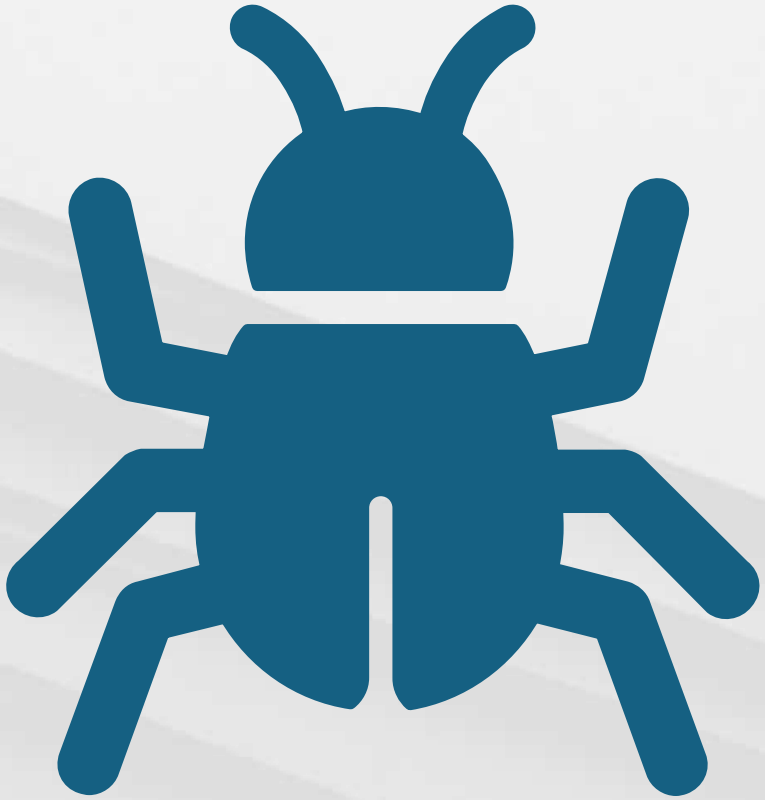
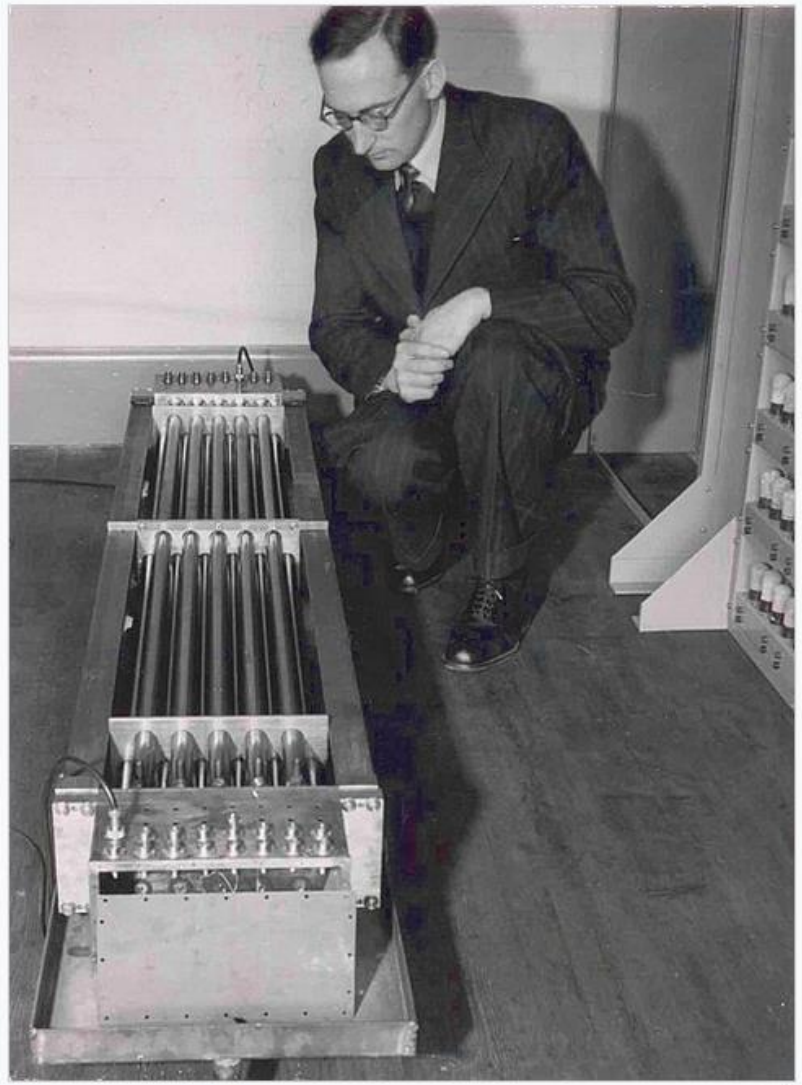
Błędy

Na podstawie: Bjarne Stroustrup – Programowanie. Teoria i praktyka z wykorzystaniem C++



„Uświadomiłem sobie, że od tej pory większą część mojego życia będę spędzał na wyszukiwaniu i korygowaniu własnych błędów.”

Maurice Wilkes, 1949



Maurice Wilkes inspecting the mercury delay line of the EDSAC in construction

Rodzaje błędów

- Błędy kompilacji (Compile-time errors) – błędy znalezione przez kompilator
  - Błędy składni (Syntax errors)
  - Błędy typów (Type error)
- Błędy konsolidacji (Link-time errors) – błędy napotkane przez konsolidator (linker)
- Błędy działania (Run-time errors) – błędy wykrywane przez mechanizmy sprawdzające w czasie działania programu
- Błędy logiczne (Logic errors) – błędy znalezione przez programistę szukającego powodów błędnych wyników

Źródła błędów

- Słaba specyfikacja – jeśli nie wiemy dokładnie, co program ma robić, jest mało prawdopodobne, że zapewnimy obsługę wszystkich możliwych przypadków, tzn. że dla każdych danych wejściowych program zwróci poprawną odpowiedź lub wyświetli komunikat o błędzie
- Niekompletne programy – w czasie prac nad programem są oczywiście rzeczy, którymi się jeszcze nie zajęliśmy
- Nieoczekiwane argumenty – np. wywołanie funkcji `sqrt(-1.2)`

Źródła błędów

- Niespodziewane dane wejściowe – programy zazwyczaj wczytują dane (z klawiatury, pliku, GUI, sieci itp.)
- Niespodziewany stan – dane (stan), które przechowuje program są niepełne lub błędne
- Błędy logiczne – kod, który nie robi tego, co zaplanował programista

PRZYKŁAD

```
int area(int length, int width) { // calculate area of a rectangle
    return length*width;
}

int framed_area(int x, int y)  { //calculate area within frame
    return area(x-2,y-2);
}

int main() {
    int x = -1;
    int y = 2;
    int z = 4;
    int area1 = area(x,y);
    int area2 = framed_area(1,z);
    int area3 = framed_area(y,z);
    double ratio = double(area1)/area3;
}
```

Obserwacje i pytania

- Obliczone wartości area1 oraz area2 są ujemne
- Czy powinniśmy zaakceptować takie wartości?
- Jeśli nie, to kto powinien wykryć te błędy – ten kto wywołał funkcję area(), czy sama funkcja?
- Przy obliczaniu area3 – dzielenie przez 0

Alternatywa

Rozwiążemy problem błędnych argumentów funkcji area()
Mamy dwie możliwości:

- Niech wywołujący funkcję area() poradzi sobie z tymi argumentami
- Niech wywołany, tzn. funkcja area(), poradzi sobie z tymi argumentami

Rozwiązanie problemu przez wywołującego

Trzeba by pisać przy każdym wywołaniu funkcji area() coś w stylu:

```
if (x<=0) error("non-positive x");
if (y<=0) error("non-positive y");
int area1 = area(x,y);
```

Rozwiązanie problemu przez wywołującego

Musimy też obsłużyć wywołania funkcji area(), które znajdują się wewnątrz funkcji framed_area():

```
if (z<=2)
    error("non-positive 2nd area() argument called by framed_area()");
int area2 = framed_area(1,z);
if (y<=2 || z<=2)
    error("non-positive area() argument called by framed_area()");
int area3 = framed_area(y,z);
```

Rozwiązanie problemu przez wywołującego

```
if (z<=2)
    error("non-positive 2nd area() argument called by framed_area()");
int area2 = framed_area(1,z);
if (y<=2 || z<=2)
    error("non-positive area() argument called by framed_area()");
int area3 = framed_area(y,z);
```

Kod jest brzydki, ale problem jest większy.

Żeby napisać go napisać musimy wiedzieć w jaki sposób funkcja framed_area() wykorzystuje funkcję area() !!!

Rozwiązanie problemu przez wywoływanego

```
int area(int length, int width) // calculate area of a rectangle
{
    if (length<=0 || width <=0) error("non-positive area() argument");
    return length*width;
}
```

To wystarczy !!!

Morał

Sprawdzaj argumenty w funkcji,
chyba że masz bardzo dobry powód,
aby tego nie robić.

Co robić, gdy znajdziemy błąd?

- Przerwać działanie programu (brutalne)
- Zwrócić specjalną wartość (nie zawsze jest kandydat na tę wartość)
- Ustawić kod błędu (wywołujący rzadko sprawdza kod błędu)
- WYJĄTKI (Exceptions) (zalecana technika)

Wyjątki

Jeśli funkcja znajdzie błąd, którego nie potrafi obsłużyć, nie kończy działania normalnie (nie wykonuje **return**)

Zamiast tego zgłasza wyjątek (**throw**) informujący, co się stało.

Każdy bezpośredni lub pośredni wywołujący może ten wyjątek przechwycić (**catch**), tzn. określić co robić w tej sytuacji.

Jeżeli żaden wywołujący nie obsłuży wyjątku, program zakończy działanie.

Hello Exceptions I

```
#include <iostream>
using namespace std;

class Bad_area { }; // a type specifically for reporting errors from area()

// calculate area of a rectangle;
// throw a Bad_area exception in case of a bad argument
int area(int length, int width)
{
    if (length<=0 || width<=0) throw Bad_area{};
    return length*width;
}


int framed_area(int x, int y) // calculate area within frame
{
    return area(x - 2, y - 2);
}
```

Hello Exceptions II

```
int main()
{
    try
    {
        int x = -1;
        int y = 2;
        int z = 4;
        int area1 = area(x, y);
        int area2 = framed_area(1, z);
        int area3 = framed_area(y, z);
        double ratio = area1 / area3;
    }
    catch (Bad_area)
    {
        cout << "Oops! bad arguments to area()\n";
    }
}
```