Lista 11 – Wyjątki

1. Napisz prosty kalkulator, który będzie wczytywał z wiersza poleceń (*command line*) liczbę całkowitą, znak definiujacy operator C++ oraz kolejną liczbę całkowitą, po czym wyświetli wynik operacji. Znak oddzielony jest od liczb dowolną liczbą białych znaków (spacja, tabulator, etc.) i przyjmuje jedną z wartości: `'+'`, `'-'`, `'/'`, `'*'`, `'%'`, `'&'`, `'|'`, `'^'`. Operacja wykonywana na wczytanych liczbach odpowiada operatorowi C++ zapisywanemu tym znakiem. Program powinien być zabezpieczony przed wprowadzeniem niepoprawnych danych: np. napisu zamiast liczby, użycia znaku spoza powyższego zbioru, dzielenia przez zero.

   Warunki brzegowe:

   a) Całą logikę programu umieść w osobnej funkcji

   ```
   void run(int argc, const char* argv[])
   ```

   b) Funkcja **`run`** na wszelkie błędy reaguje zgłoszeniem wyjątku (np. std::logic_error) ze stosownym komunikatem diagnostycznym

   c) Funkcja **`main`** wywołuje funkcję **`run`** oraz obsługuje wyjątki, i nic więcej

   d) Do zmiany napisów (wczytywanych z wiersza poleceń) na liczby użyj funkcji **`std::stoi`**. Pamiętaj, że może ona zgłosić wyjątek.

   e) Do obsługi znaków (operatorów) w funkcji **`run`** zastosuj instrukcję **`switch`**.

   Uwaga. Wprowadzenie z linii komend niektórych znaków, zwłaszcza `&` i `|`, może być kłopotliwe. U mnie działa poprzedzenie ich ukośnikiem (np.: `6 \& 8`) lub ujęcie w znaki cudzysłowu (np.: `7 "|" 9`).

Pytania kontrolne

1. Wyjątki w C++:
   a) używane są jako wygodna alternatywa dla instrukcji sterujących (np. `if`, `switch`, `goto`)
   b) używane są do sygnalizacji sytuacji/stanów, których nie można obsłużyć w miejscu ich detekcji (program wszedł w ślepą uliczkę i musi się wycofać, by móc poprawnie kontynuować działanie)
   c) są aktywne wyłącznie w trybie `Release`;
2. Blok `try/catch` służy do:
   a) zgłaszania wyjątków
   b) wyłapywania i obsługi wyjątków
   c) debugowania programów
3. Wyjątkiem w C++ może być
   a) dowolny obiekt lub zmienna
   b) wyłącznie obiekt klasy `std::exception` lub jej klasy pochodnej
   c) wyłącznie wskaźnik lub referencja do obiektu globalnego
4. Skoro klasa `std::exception` posiada wirtualną stałą metodę `what()`, to wyjątki tej klasy lub jej klas pochodnych zaleca się wyłapywać
   a) w bloku `catch(std::exception e)`
   b) w bloku `catch(const std::exception & e)`
   c) w bloku `catch(...)`
5. Wyjątki w C++ zgłasza się instrukcją:
   a) `catch`
   b) `throw`
   c) `try`

6. Niewyłapany wyjątek:
   a) przepada (jest ignorowany)
   b) powoduje natychmiastowe zakończenie funkcji, w której został zgłoszony, po czym zgłaszany jest w funkcji nadrzędnej
   c) powoduje zakończenie programu, jeśli nie zostanie wyłapany w żadnej funkcji

7. RAII:
   a) to temat jednego z pytań na egzaminie inżynierskim
   b) to zasada mówiąca o tym, w jaki sposób w C++ można pogodzić mechanizm wyjątków z potrzebą zapewnienia bezpieczeństwa zasobom – bez korzystania z automatycznej śmieciarki (*garbage collector*)
   c) to zasada opisująca zalecany styl programowania (sposób posługiwania się językiem przez programistów) a nie element języka czy biblioteki standardowej

8. Zasada RAII mówi, że
   a) Czas użytkowania każdego zasobu, który przed pierwszym użyciem musi być zdobyty, należy związać z czasem życia pewnego obiektu (*a C++ programming technique which binds the life cycle of a resource that must be acquired before use to the lifetime of an object*)
   b) Zasoby należy do programu przydzielać wyłącznie w konstruktorze, a zwalniać – wyłącznie w destruktorze
   c) Nie wolno przydzielać do programu zasobów w tych fragmentach kodu, które mogą zgłosić wyjątek

9. Zasada RAII mówi, że
   a) Zarządzanie każdym zasobem należy zdefiniować w osobnej klasie odpowiadającej temu zasobowi
   b) Zasoby należy zdobywać w konstruktorze. W przypadku niepowodzenia, konstruktor powinien zgłosić wyjątek
   c) Zasoby należy zwalniać w destruktorze. Destruktor w żadnym wypadku nie może zgłosić wyjątku.

10. Następujące wzorce kodu nie są zgodnie (ściśle) z zasadą RAII. W których z nich mimo tego pewien zasób (tu: plik) jest zarządzany poprawnie nawet w przypadku zgłoszenia wyjątku w kodzie zastąpionym wielokropkiem (. . .):

a)
```
std::ifstream in("plik.txt");
...
```

b)
```
FILE * in = fopen("plik.txt", "r");
...
fclose(in);
```

c)
```
std::ifstream in;
...
in.open("plik.txt");
...
in.close();
```

11. Zwijanie stosu (*stack unwinding*):
   a) następuje po napotkaniu klamry zamykającej zakres (`}`), po zakończeniu funkcji instrukcją `return` lub po zgłoszeniu wyjątku
   b) polega na automatycznym wywołaniu destruktorów obiektów automatycznych w danym zakresie (lub w funkcji) w kolejności odwrotnej do kolejności ich konstrukcji
   c) to algorytm dostępny dla kontenera `std::stack<T>`

12. Przykładami zasobów, o których mówi zasada RAII, są:
   a) pamięć na stercie, wątki, gniazda sieciowe, otwarte pliki, muteksy i inne mechanizmy synchronizacji wątków, połączenia z bazami danych
   b) pamięć na stosie, czas procesora, dostępne rdzenie procesora, dostępna pamięć cache, przepustowość sieci, prąd zasilający komputer, zbiór instrukcji procesora, zbiór dostępnych funkcji systemowych, przepustowość i liczba kanałów pamięci RAM.

13. Wybierz najlepszą kolejność obsługi wyjątków:
   a) `catch (const std::exception & e) {}` `catch (const std::logic_error & e) {}` `catch (...) {}`
   b) `catch (...){}` `catch (const std::exception & e) {}` `catch (const std::logic_error & e) {}`
   c) `catch (const std::logic_error & e) {}` `catch (const std::exception & e) {}` `catch (...) {}`