**POLITECHNIKA KRAKOWSKA**
WYDZIAŁ INŻYNIERII ELEKTRYCZNEJ I KOMPUTEROWEJ

# Inteligentne wskaźniki

## Systemy odporne na błędy

Sprawozdanie z laboratorium nr 5

Maria Guz
Karol Pątko
Wojciech Maludziński

# Wstęp

Inteligentne wskaźniki to abstrakcyjny typ danych symulujący wskaźnik, dodając przy tym nowe funkcje takie jak odśmiecanie albo sprawdzanie zakresów odwołań (bounds checking). Ich charakterystyczną cechą jest automatyczne niszczenie zawartości w chwili, kiedy przestaje być użyteczny. Dzięki wyborze odpowiedniego typu wskaźnika mamy kontrolę nad czasem życia przechowywanej wartości. Inteligentne wskaźniki są typami szablonowymi - można w nich przechowywać wartość dowolnego typu. Można zainicjalizować inteligentny wskaźnik poprzez jawne użycie operatora *new* lub wykorzystanie funkcji *std::make_unique*. W języku C++ inteligentne wskaźniki mogą zostać zaimplementowane jako wzorzec klasy, który dzięki przeciążeniu operatorów, udaje działanie zwykłego wskaźnika definiując dodatkowe algorytmy zarządzania pamięcią.

Przykłady inteligentnych wskaźników:

- *unique_ptr* – pozwala na zdefiniowanie dokładnie jednego obiektu opakowującego zwykły wskaźnik.
- *shared_ptr* - klasa inteligentnego wskaźnika zliczająca ilość odwołań do niego z różnych miejsc programu. Używa się jej gdy zwykły wskaźnik musi być widoczny z wielu miejsc w kodzie, na przykład gdy zwraca się kopię wskaźnika z pojemnika, ale należy zachować oryginał.
- *weak_ptr* – specjalny rodzaj używany łącznie z klasą *shared_ptr*. Zapewnia dostęp do obiektu identyfikowanego przez jeden lub więcej obiektów *shared_ptr*, bez związku z ilością odwołań.

# Cel zajęć

Zapoznanie studentów z inteligentnymi wskaźnikami oraz ich implementacja w języku C++.

# Treść zadania

*Napisz program, który stworzy serię rekordów, a następnie usunie losowo wybrany ze sprawdzaniem zajętej pamięci.*

# Kod programu

*main.cpp*

```
#include <memory>
#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;

#define NUMBER_OF_POINTERS 20

void print(shared_ptr<double>* pointers){
   for (int i = 0; i < NUMBER_OF_POINTERS; ++i) {
       printf("[%d] %f\n", i, *pointers[i]);
   }
   cout << endl;
}

void print(weak_ptr<double>* weakPointers) {
   shared_ptr<double> val;
   for (int i = 0; i < NUMBER_OF_POINTERS;  ++i) {
       if (weakPointers[i].lock()) {
           printf("[%d] %f\n", i, *weakPointers[i].lock());
       }
       else {
           printf("[%d] usunieta\n", i);
       }
   }
}

float randFloat(int a, int b){
   return (1.0 * rand() / (RAND_MAX)) * (b - a) + a;
}

void deleteRandomValue(shared_ptr<double>* pointers) {
   int randPointer = rand() % NUMBER_OF_POINTERS;
   pointers[randPointer].~shared_ptr();
}

int main(void) {
   shared_ptr<double> pointers[NUMBER_OF_POINTERS];
   weak_ptr<double> weakPointers[NUMBER_OF_POINTERS];

   srand(time(NULL));
   float randomValue;
   for (int i = 0; i < NUMBER_OF_POINTERS; ++i) {
       // losuje wartosci z przedzialu [-25, 25]
       randomValue = randFloat(-25, 25);

       //zapisuje wartosci do wskaznika
       pointers[i] = make_shared<double>(randomValue);

       // "obserwator" wskaznikow pointers
       weakPointers[i] = pointers[i];
   }

   //wypisuje wylosowane liczby
   print(pointers);

   //usuwam losowa wartosc
   deleteRandomValue(pointers);

   //wypisuje wartosci z oznaczeniem usunietej
   print(weakPointers);

   return 0;
}
```

W pliku *main.cpp* zawiera się całość programu. Zdefiniowane zostały funkcje do wyświetlania dwóch rodzajów wskaźników, do losowania liczby, i do usuwania losowej

wartości spośród wskaźników. W funkcji *main()*, po deklaracji dwóch tablic wskaźników oraz pozostałych zmiennych, w pętli losujemy zawartości, zapisujemy do tablic wskaźników, oraz nadajemy obserwatorów. Po wyjściu z pętli wyświetlamy tablicę wskaźników, kasujemy losową wartość, i wyświetlamy tablicę obserwatorów wskaźników.

# Działanie programu

```
[0] -10.547978
[1] 21.745726
[2] -22.128422
[3] -13.792480
[4] 14.805937
[5] -3.389609
[6] -1.055086
[7] -8.332948
[8] -15.120472
[9] 19.427803
[10] -22.596752
[11] -3.978986
[12] -22.401812
[13] -0.200137
[14] 15.433125
[15] -24.401390
[16] -12.498335
[17] 14.341691
[18] 10.098504
[19] 9.429403

[0] -10.547978
[1] 21.745726
[2] -22.128422
[3] -13.792480
[4] 14.805937
[5] -3.389609
[6] -1.055086
[7] usunieta
[8] -15.120472
[9] 19.427803
[10] -22.596752
[11] -3.978986
[12] -22.401812
[13] -0.200137
[14] 15.433125
[15] -24.401390
[16] -12.498335
[17] 14.341691
[18] 10.098504
[19] 9.429403
```

# Wnioski

Inteligentne wskaźniki mogą służyć do opakowania wskaźników w obiekt. Przez wykorzystanie *weak_ptr* (słabych wskaźników), możemy je obserwować bez wpływania na ich zawartość, z kolei *shared_ptr* pozwoliłoby na zliczanie ilości odwołań do wskaźników. Inteligentne wskaźniki są przydatnym narzędziem, zapobiegają wyciekom pamięci.