Krzysztof Wołkonowski

Zadanie 1

Alokujemy pamięć słowem kluczowym **new**, której później nie zwalniamy.
Rozwiązaniem tego problemu jest ręczne zwolnienie tej pamięci przez polecenie słowem kluczowym delete.

Zadanie 2

Przy pomocy funkcji allocateInts() tworzymy wektor z dynamicznie zaalokowanymi intami. W kodzie programu mamy funkcję deallocateInts(), która przyjmuje za argument wektor z intami do zwolnienia. Jednak nie wołamy tej funkcji wewnątrz funkcji main().
Rozwiązaniem tego problemu jest wywołać funkcję deallocateInts() wewnątrz main() z wektorem num jako argumentem.

Zadanie 3

W funkcji main() dynamicznie tworzymy instancję klasy Resource, jednak robimy to w bloku try. Uniemożliwia nam to zwolnienie tej pamięci. Zauważmy, że jedyną ryzykowną instrukcją jest metoda use(). Wyciągając polecenia new i delete spoza bloku try, mamy możliwość zwolnienia tej pamięci nawet gdy wystąpi wyjątek. Rozwiązuje to nasz problem. Warto również zauważyć, że jedynie przekazanie ‘d’ jako pierwszy argument wywołania programu powoduje wystąpienie wyjątku.
Chcąc utworzyć własny wyjątek, tworzymy klasę dziedziczącą po podanym std::logic_error. Tworzymy dla niej konstruktor, który wywołuje najpierw konstruktor z argumentem klasy pierwotnej. Zaznaczamy, że będziemy wykorzystywać funkcję what() z klasy pierwotnej poleceniem using.

Zadanie 4

Konstruktor klasy MyPointer dynamicznie alokuje pamięć, a jednocześnie wywołuje funkcję zrzucającą wyjątek. Obiekt jest w pełni utworzony jedynie, gdy wykona się cały konstruktor, co się nie dzieje w tym przykładzie. Oznacza to, że nie wykona się również destruktor zwalniający pamięć.
Rozwiązaniem tego problemu jest otoczenie całego konstruktora przez blok try-catch, gdzie w przypadku wystąpienia wyjątku pamięć będzie zwalniana.

Zadanie 5

Instancje klasy Partner są dynamicznie alokowane w funkcji convertMe(), natomiast blok try-catch znajduje się w funkcji main(). W efekcie, gdy wystąpi wyjątek, pamięć nie zostanie zwolniona.
Zastępujemy w kodzie zwykłe wskaźniki przez unique_ptr. Możemy utworzyć je funkcją make_unique<Partner>(). Nie potrzebujemy wtedy poleceń delete. Gdy wystąpi wyjątek, odwijanie stosu wywoła destruktor unique_ptr (zaalokowanego na stosie, nie stercie), który usunie dynamicznie zaalokowaną pamięć na instancję klasy Partner.

Zadanie 6

Przy zapisie do pliku najpierw upewniamy się, że plik został już zainicjowany. Następnie wywołujemy funkcję fprintf, do której musimy przekazać zwykły wskaźnik zamiast shared_ptr. std::string również zamieniamy na C-string metodą c_str().
Przy funkcji makeFile() musimy utworzyć shared_ptr do zadanego pliku. Mamy zadaną nazwę pliku i flagi do jego otwarcia, zatem możemy wywołać funkcję fopen z tymi argumentami. Otrzymujemy wtedy wskaźnik, zatem możemy wywołać shared_ptr<std::FILE> od tego wskaźnika. Musimy jednak zadbać o zamknięcie pliku na koniec wypisywania. Posłuży nam do tego deleter, przekazany jako drugi argument konstruktora shared_ptr. Zawieramy w nim funkcję fclose, aby zamknąć zasób.