

Mikroelektronika w Technice i Medycynie

"Podstaw programowania obiektowego"

Ćwiczenia laboratoryjne

Instrukcja do kolokwium z C++

Konwencje

1. Obiekty powinny nazywać się tak jak ich klasy plus przedrostek „My”
2. Każdy plik .h powinien zawierać tzw. „Include guard” (patrz Wikipedia)

Organizacja

1. Kolokwium będzie trwać 1.5h
2. Refaktoryzacja kodu będzie odbywać się z użyciem środowiska Keil4
3. Na pulpicie, w katalogu „PPO”, znajdować się będą wszystkie potrzebne pliki, tzn. niniejsza instrukcja oraz katalog z projektem do refaktoryzacji (nazwa „1”)
4. W trakcie kolokwium
 - 4.1. Na pulpicie stworzyć katalog ze swoim nazwiskiem i imieniem
(nazwisko najpierw, bez polskich liter, przykład „MiroslawZoladz”)
 - 4.2. Do tego katalogu skopiować katalog z dostarczonym kodem źródłowym
 - 4.3. Następnie wykonywać poszczególne kroki refaktoryzacji od kroku 2. Dla każdego kroku:
 - 4.3.1. Zamknąć Keila (jeśli był otwarty)
 - 4.3.2. Zrobić kopię katalogu z poprzednim krokiem nadając mu nazwę aktualnego kroku
 - 4.3.3. Wykonać refaktoryzację zgodnie z instrukcją
5. Po zakończeniu kolokwium należy:
 - 5.1. Spakować katalog ze swoim nazwiskiem do pliku .zip,
 - 5.2. Poprosić prowadzącego o odebranie pliku .zip
 - 5.3. Poprosić prowadzącego o podłączenie internetu w celu pobrania pliku zip „dla siebie” (opcjonalnie)
 - 5.4. Wyłączyć komputer

Zasady oceniania

1. Kolokwium oceniane jest w systemie zerojedynkowym (zaliczone/niezaliczone).
2. Przyczyna nieuzyskania zaliczenia może być:
 - Niewykonanie któregoś z podpunktów w ogóle lub zgodnie z instrukcją lub błędy kompilacji
 - 5-krotne (sumarycznie) niedostosowanie się do któregoś z poniższych punktów:
 - Nazwy plików powinny być zgodne z nazwami klas, czyli: led, ledpos, ledneg, keyboard, stepper.
 - **Pliki nie powinny zawierać żadnych zbędnych "includów".**
 - Dostęp do pól i metod oraz klas bazowych powinien być:
 - maksymalnie ograniczony
 - określony explicite (nie domyślny).
 - Program nie powinien posiadać żadnych zbędnych, tj. nieużywanych elementów ani komentarzy.
 - Jeżeli używany jest wskaźnik do klasy bazowej, to nie należy dołączać do pliku, w którym jest on używany plików nagłówkowych z definicją klas dziedziczących z tej klasy (bazowej).
 - Metody wirtualne należy określać explicite.

Kroki refaktoryzacji

1. Przygotowanie katalogu projektu

Opisano w 4.1-2

2. Zmienić moduł `Led` na klasę:

W pliku `.h` zadeklarować klasę `Led`.

Zmienić funkcje na metody klasy.

Zmienną do liczenia kroków uczynić składnikiem klasy.

Metody klasy `Led` zdefiniować w pliku `.cpp`.

Stworzyć obiekt klasy `Led` w pliku `main.cpp`.

Zmodyfikować `main.cpp` tak aby program działał jak w poprzednim punkcie.

Zarchiwizować stan projektu.

Stepper
-LedCtr : unsigned char
+Init() +StepLeft() +StepRight() -Step() -On()

3. Wyodrębnić z klasy `Led` klasę `Stepper`:

Klasa `Stepper` powinna być zdefiniowana w oddzielnych plikach `stepper.h` i `stepper.c`. Powinna składać się z metod `StepLeft`, `StepRight` i `Step`.

W `main.cpp` stworzyć obiekty globalne klasy `Led` i `Stepper`.

Przetestować i zarchiwizować projekt.

Led
+Init() +On()

Stepper
-LedCtr : unsigned char
+StepLeft() +StepRight() -Step()

4. Przenieść inicjalizację portów klasy `Led` do konstruktora (domniemanego):

Zastąpić metodę `Init` klasy `Led` konstruktorem domniemanym.

Doprowadzić program do działania.

Sprawdzić symulatorem, kiedy uruchamiany jest konstruktor `Led`.

Przetestować i zarchiwizować projekt.

Led
+Led() +On()

Stepper
-LedCtr : unsigned char
+StepLeft() +StepRight() -Step()

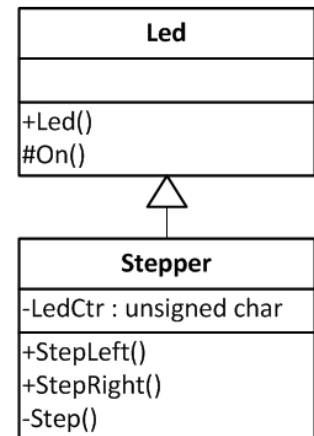
5. Zamiast użycia globalnego obiektu klasy `Led` użyć mechanizmu dziedziczenia:

Usunąć obiekt globalny klasy `Led`.

Ustawić dziedziczenie klasy `Stepper` z klasy `Led`.

UWAGA: Ograniczyć dostęp do składników `Led` do minimum.

Przetestować i zarchiwizować projekt.



6. Użyć konstruktora klasy `Stepper` do ustawiania początkowej pozycji `Led-a`:

Zmniejszyć częstotliwość pętli głównej do 2 Hz.

Użyć konstruktora domniemanego klasy `Stepper` do ustalenia początkowej pozycji `Led-a` na 2 (licząc od 0).

Doprowadzić program do działania. Przetestować.

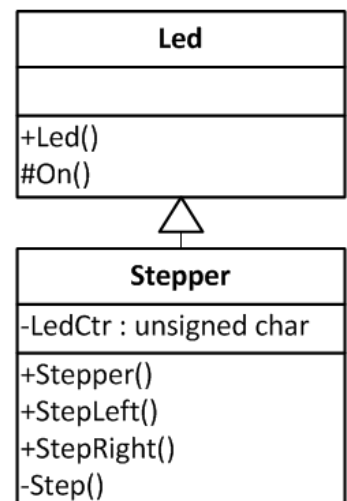
Sprawdzić symulatorem, w jakiej kolejności uruchamiane są konstruktory obiektów klas `Led` i `Stepper`.

Zastąpić konstruktor domniemany konstruktorem `Stepper(unsigned char)`, który pozwoli na ustalenie dowolnej początkowej pozycji `Led-a`.

Ustawić wartość domyślną parametru konstruktora na 0.

Doprowadzić program do działania.

Przetestować i zarchiwizować projekt.



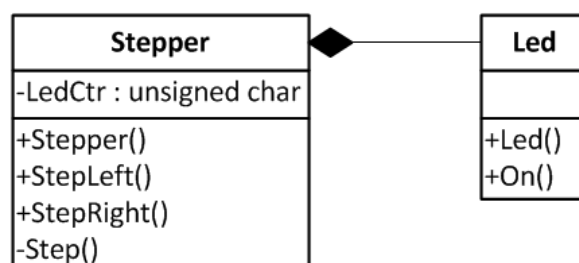
7. Zastąpić dziedziczenie kompozycją.

Usunąć z klasy `Stepper` dziedziczenie z klasy `Led`.

Uczynić obiekt klasy `Led` elementem składowym klasy `Stepper` („`MyLed`”).

Sprawdzić symulatorem, w jakiej kolejności uruchamiane są konstruktory obiektów klas `Led` i `Stepper`.

Przetestować i zarchiwizować projekt.



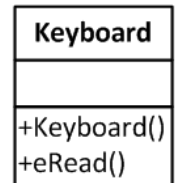
8. Konfiguracja programu do trybu inwersji ledów

Opis działania: Jeżeli w trakcie resetu BUTTON_4 był wciśnięty, to stan ledów powinien być zanegowany (chodzące zero zamiast jedynki)

8 a) Przygotowanie

Usunąć konstruktor klasy Stepper.

Dodać moduł `keyboard` do projektu (skopiować do katalogu projektu pliki `.cpp` i `.h`), a następnie przerobić go na klasę `Keyboard` wg. rysunku widocznego obok.



W funkcji `main` stworzyć obiekt klasy `Keyboard`.

Zmodyfikować pętlę główną tak, aby punkt świetlny przesuwiał się w prawo jeśli naciśnięto BUTTON_1, w lewo jeśli naciśnięto BUTTON_2 oraz nie przesuwiał wcale jeśli nie naciśnięto żadnego przycisku.

Przetestować i zarchiwizować projekt.

8 b) Użycie instrukcji wyboru

Dodać zmienną globalną (`ucInversion`), która będzie przechowywać informacje o naciśnięciu przycisku BUTTON_4 podczas resetu.

Zmodyfikować metodę `On` klasy `Led` tak, aby w zależności od stanu BUTTON_4 podczas resetu zapalała lub gasiła led.

Przetestować i zarchiwizować projekt.

8 c) Wyodrębnienie z klasy `Led` klasę `LedInv`

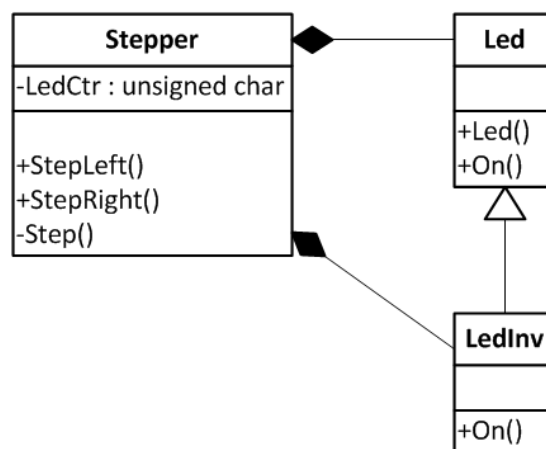
Stworzyć klasę `LedInv` (oddzielne pliki), dziedziczącą z klasy `Led`, której metoda `On` będzie gasić wybrany led.

W klasie `Led` w metodzie `On` pozostawić jedynie możliwość zapalenia jednego led.

Dodać obiekt klasy `LedInv` do składników klasy `Stepper`.

Zmodyfikować metodę `Step` tak, aby program realizował funkcjonalność z poprzedniego punktu.

Przetestować i zarchiwizować projekt.



8 d) Usunięcie z klasy **Stepper** odwołania do zmiennej globalnej

Usunąć zmienną globalną `ucInversion`.

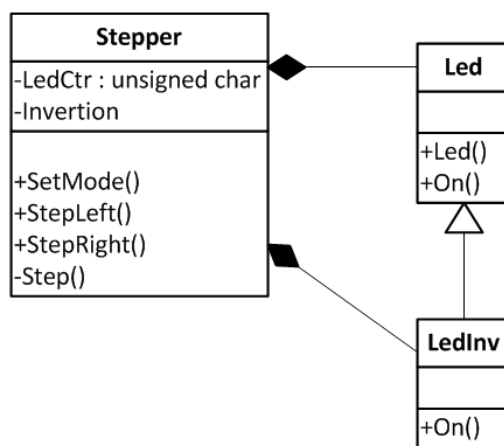
Dodać do klasy **Stepper** składową `ucInversion`.

Dodać metodę `SetMode(unsigned char)` pozwalającą na ustawienie składowej `ucInversion`.

Zmodyfikować `main.cpp` tak, aby program realizował funkcjonalność z poprzedniego punktu.

Sprawdzić ile razy uruchamia się konstruktor klasy **Led**.

Przetestować i zarchiwizować projekt.



8 e) Usunięcie instrukcji wyboru z klasy **Stepper** (polimorfizm)

Usunąć z klasy **Stepper** składnik `MyLedInv`.

Zamienić składnik `MyLed` na wskaźnik na obiekt typu `Led` (`pMyLed`).

Dodać do zmiennych funkcji `main` obiekty klasy `Led` i `LedInv`.

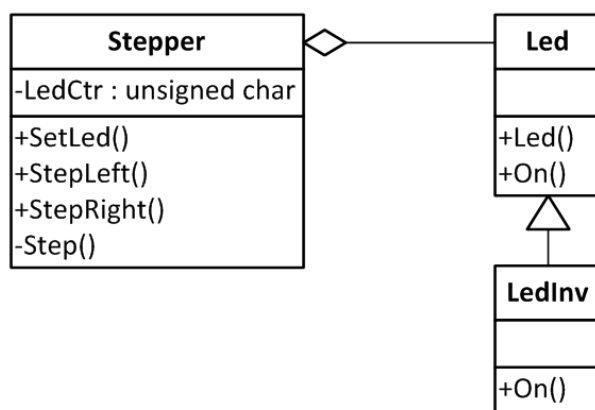
W miejsce metody `SetMode` stworzyć metodę `SetLed`, która pozwala na ustawianie składowej `pMyLed`.

Odpowiednio zmodyfikować metodę `Step`.

Zmodyfikować `main.cpp` tak, aby program realizował funkcjonalność z poprzedniego punktu.

UWAGA: zastosować polimorfizm.

Przetestować i zarchiwizować projekt.



8 f) Zastosowanie metody czysto wirtualnej

Dodać klasę `LedPos` dziedziczącą z klasy `Led`.

Przenieść implementację metody `On` z klasy `Led` do `LedPos`,

Metodę `On` w klasie `Led` pozostawić całkowicie wirtualną („...`On(unsigned char) = 0`”).

Zmodyfikować `main.cpp` tak, aby program realizował funkcjonalność z poprzedniego punktu.

Przetestować i zarchiwizować projekt.

