Mikroelektronika w Technice i Medycynie
Luty 2017

Instrukcja do ćwiczeń/kolokwium z C++

Zakres wymaganych wiadomości

„Symfonia c++”, rozdziały: 10.1-14, 14.1-6, 19.1-6, 19.9, 20

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. Stworzyć na pulpicie **katalog główny** *NazwiskoImie* (wpisać swoje nazwisko i imię).
2. Prowadzący dostarczy pendrive zawierający:
   2.1. instrukcją do kolokwium,
   2.2. katalog *Sources*, zawierający pliki źródłowe.
3. Zawartość pendrive należy niezwłocznie skopiować do stworzonego uprzednio katalogu.
4. Wymontować pendrive i zwrócić prowadzącemu.
5. W katalogu głównym utworzyć **katalog projektu** środowiska Keil i nazwać go *Project*.
6. Archiwizacja projektu polega na skopiowaniu całego katalogu projektu i nadaniu mu nazwy odpowiadającej numerowi ukończonego ćwiczenia (np. 2 lub 8a).
7. **Przez cały czas należy pracować wyłącznie na projekcie z katalogu *Project*, nie na katalogach zarchiwizowanych. Zalecane jest nie zamykać środowiska Keil.**
8. Po zakończeniu kolokwium należy:
   8.1. spakować katalog główny do pliku .zip,
   8.2. podpiąć pendriva i skopiować na niego plik .zip zawierający katalog główny,
   8.3. wymontować pendriva,
   8.4. poczekać na potwierdzenie przez prowadzącego zawartości pendriva,
   8.5. wyłączyć komputer.

Zasady oceniania

1. Kolokwium oceniane jest w systemie zerojedynkowym (zaliczone/niezaliczone).
2. Przyczyna nieuzyskania zaliczenia może być:
   2.1. niewykonanie któregoś z podpunktów,
   2.2. 4-krotne (sumarycznie) niedostosowanie się do któregoś z poniższych punktów:
      2.2.1. Nie należy usuwać katalogów ani zmieniać ich nazw.
      2.2.2. Nazwy plików powinny być zgodne z nazwami klas, czyli: led, ledpos, ledneg, keyboard, stepper.
      2.2.3. Pliki nie powinny zawierać **żadnych** zbędnych "includów".
      2.2.4. Dostęp do pól i metod oraz klas bazowych powinien być jak najbardziej ograniczony i określony explicite (nie domyślnie).
      2.2.5. Program nie powinien posiadać żadnych zbędnych, tj. nieużywanych elementów ani komentarzy.
      2.2.6. 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).
      2.2.7. Metody wirtualne należy określać explicite.

1. Przygotowanie projektu

Skopiować do katalogu *Project* pliki `main` oraz `led` (.cpp i .h) z katalogu *Sources*.

W katalogu stworzyć nowy projekt:

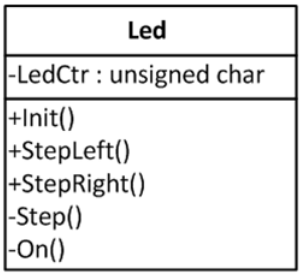
- Procesor NXP -> LPC2131/01.
- Dodać kod startowy.
- Wybrać odpowiednie ustawienia projektu (Project -> Options for Target).

Dodać do projektu („Add Existing Files …") pliki `main.cpp` oraz `led.cpp`.

Skompilować i uruchomić projekt. Poprawić ewentualne błędy.

Upewnić się, czy funkcja `Delay` przyjmuje argumenty w milisekundach. Jeśli nie, to dobrać odpowiednia stałą.

2. Zmienić moduł `Led` na klasę:



W pliku .h zadeklarować klasę `Led`.

Zmienić funkcje na metody klasy.

Zmienną do liczenia kroków uczynić składnikiem klasy.

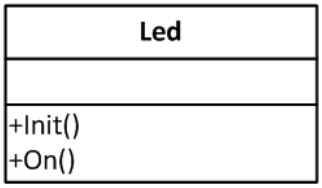
Metody klasy `Led` zdefiniować w pliku .cpp.

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

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

Zarchiwizować stan projektu.

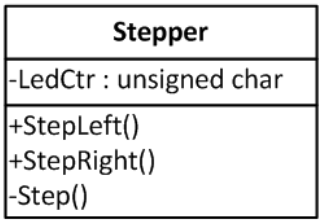
3. Wyodrębnić z klasy `Led` klasę `Stepper`:



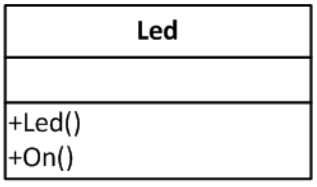
Klasa `Stepper` powinna być zdefiniowana w oddzielnych plikach `stepper.h` i `stepper.c`. Powinna składać się z metod `StepLeft`, `StepRight` i `Step`.

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

Przetestować i zarchiwizować projekt.



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

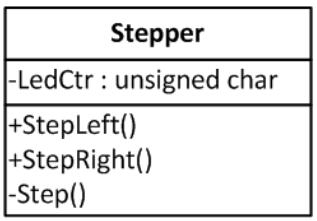


Zastąpić metodę `Init` klasy `Led` konstruktorem domniemanym.

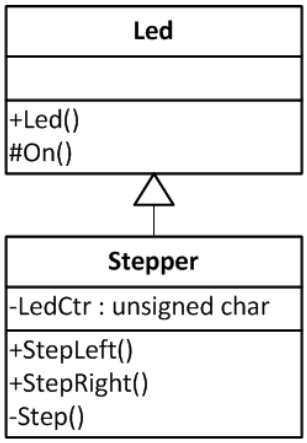
Doprowadzić program do działania.

Sprawdzić symulatorem, kiedy uruchamiany jest konstruktor `Led`.

Przetestować i zarchiwizować projekt.



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



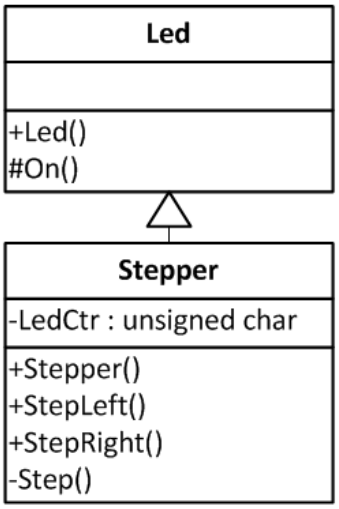
Usunąć obiekt globalny klasy `Led`.

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 ustawianie 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.

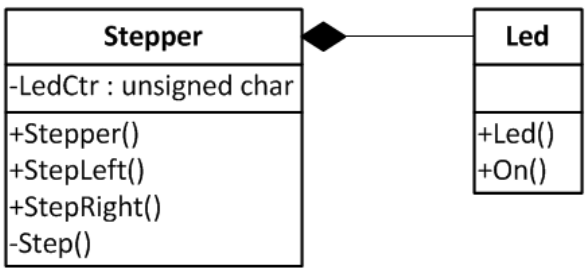
7. Zastąpić dziedziczenie kompozycją.

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`.

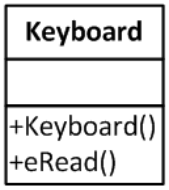
Przetestować i zarchiwizować projekt.



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 przesuwał się w prawo jeśli naciśnięto BUTTON_1, w lewo jeśli naciśnięto BUTTON_2 oraz nie przesuwał wcale jeśli nie naciśnięto żadnego przycisku.

Przetestować i zarchiwizować projekt.

8 b) Użycie instrukcji wyboru

Dodać zmienna 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 leda.

Przetestować i zarchiwizować projekt.

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

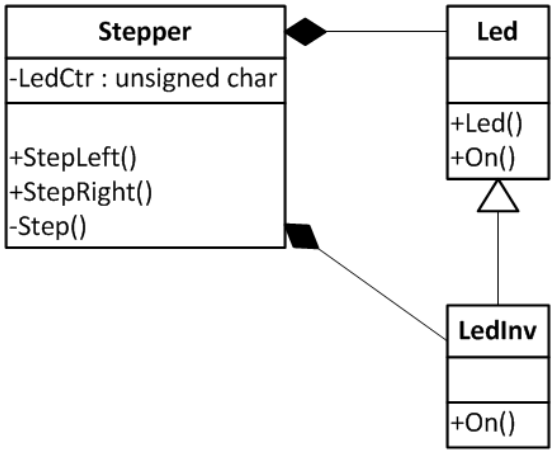
Stworzyć klasę `LedInv` (oddzielne pliki), dziedziczącą z klasy `Led`, której metoda `On` będzie gasić wybrany led.

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

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) Usuniecie z klasy `Stepper` odwołania do zmiennej globalnej

Usunąć zmienną globalną `ucInversion`.

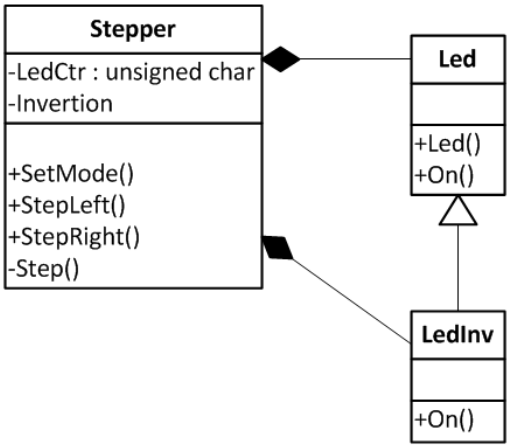
Dodać do klasy `Stepper` składową `ucInversion`.

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) Usuniecie 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`.

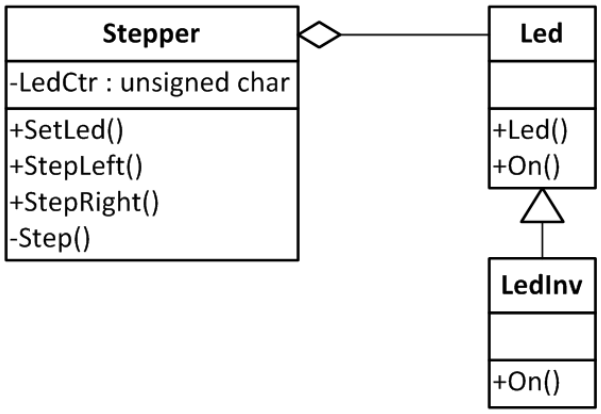
W miejsce metody `SetMode` stworzyć metodę `SetLed`, która pozwali na ustawianie składowej `pMyLed`.

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.