Zaawansowane Technologie Programowania

Projekt 2 : Zegarek z Alarmem i Timerem

Adrian Leśniak / Dominik Gajowniczek

Opis programu w języku C++.

1.Cele i założenia projektu:

- Celem projektu było stworzenie aplikacji konsolowej w języku C++, która łączy w sobie funkcjonalność zegarka, budzika i timera.
- Kluczowym założeniem było zapewnienie interaktywności z użytkownikiem poprzez komendy wiersza poleceń, co pozwala na dynamiczne zarządzanie czasem w codziennym życiu.

2.Zastosowanie projektu:

- Aplikacja może być wykorzystywana w różnych kontekstach jako narzędzie do zarządzania czasem, budzik w domu lub biurze, a także jako timer pomocny w sytuacjach wymagających odliczania czasu, takich jak gotowanie, sesje naukowe lub treningi.

3.Technologie i oprogramowanie użyte w projekcie:

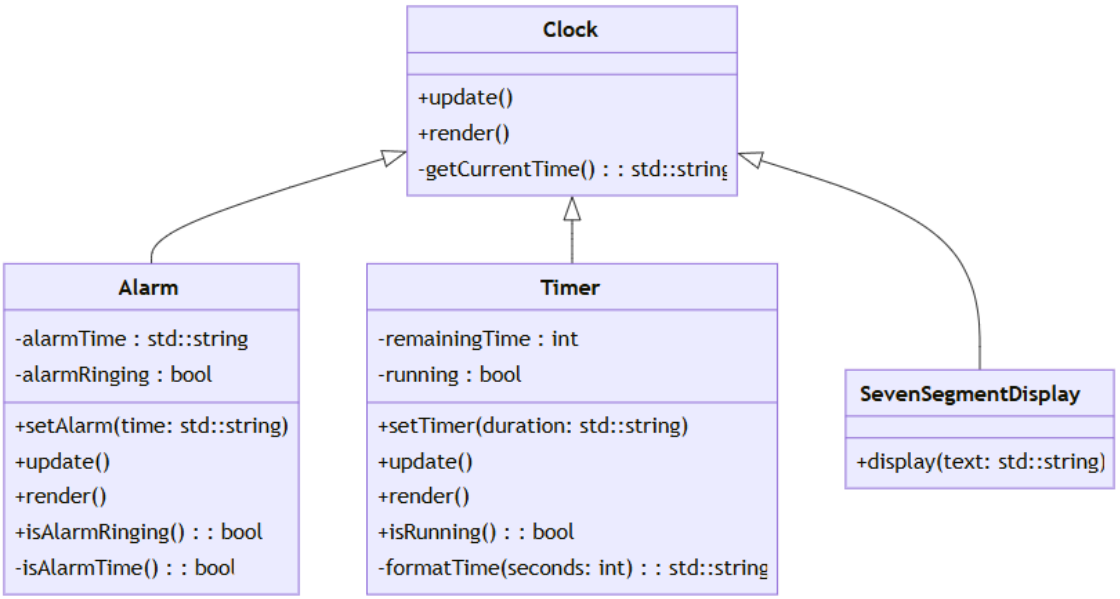
- **Język programowania:** C++
- **Środowisko programistyczne:** Visual Studio 2019, Code Blocks
- **Dodatkowe biblioteki:** Standard Template Library (STL)

4.Wymagania do uruchomienia:

- **System operacyjny:** Windows 10 lub nowszy / Linux Ubuntu 18.04 LTS lub nowszy
- **Kompilator:** g++ (GCC) wersja 8.1.0 lub nowsza / Microsoft Visual C++ 2019

5.Architektura aplikacji:

Program został zaprojektowany z modułowym podejściem, składającym się z czterech głównych klas: `Clock`, `Timer`, `Alarm`, oraz `SevenSegmentDisplay`. Każda z tych klas odpowiada za różne aspekty działania aplikacji, co pozwala na łatwe zarządzanie kodem i potencjalne rozszerzenia funkcjonalności.



Opis diagramu klas

1. Komponenty i ich relacje:
    - **`Clock`**: Centralny komponent odpowiedzialny za dostarczanie aktualnego czasu. Metoda `getCurrentTime()` jest kluczowa, ponieważ dostarcza czas, który jest wykorzystywany przez inne klasy.
    - **`Alarm`**: Zależy od `Clock` do sprawdzania, czy ustawiony czas alarmu zgadza się z bieżącym czasem. Jeśli tak, aktywuje stan alarmu poprzez `alarmRinging`.
    - **`Timer`**: Odlicza czas od zadanego momentu, opierając się na wartościach podanych w sekundach. Wykorzystuje wewnętrzny mechanizm odliczający czas, który jest aktualizowany przy każdym cyklu sprawdzania (`update()`).
    - **`SevenSegmentDisplay`**: Używana przez wszystkie trzy powyższe klasy do wyświetlania informacji w sposób, który symuluje wyświetlacz siedmiosegmentowy.

2. Metody i ich funkcje:
    - **Metody aktualizacji (`update()`)**: W każdej z klas, te metody są odpowiedzialne za cykliczne sprawdzanie stanu i przeprowadzanie niezbędnych operacji, takich jak sprawdzenie czy nadszedł czas alarmu w `Alarm` czy odliczanie czasu w `Timer`.
    - **Metody renderowania (`render()`)**: Służą do wizualizacji danych, czyli pokazania czasu, sygnału alarmowego czy odliczającego czasu timera na ekranie.

3. Interakcje między klasami:
    - Klasy `Alarm` i `Timer` obie korzystają z metody `display()` klasy `SevenSegmentDisplay` do wyświetlania odpowiednich komunikatów i danych. To pokazuje, jak klasy mogą współdzielić wspólne zasoby lub funkcjonalności.
    - Klasy te są wzajemnie odseparowane w kwestii logiki biznesowej, ale integrują się na poziomie prezentacji danych, co jest dobrym przykładem zasady jednej odpowiedzialności i reużywalności kodu.

Implikacje dla rozwoju i utrzymania

1. **Łatwość rozszerzania**: Diagram klas pokazuje, jak łatwo można by było dodać kolejne funkcje, np. dodatkowe rodzaje alarmów czy wsparcie dla różnych stref czasowych, bez konieczności ingerencji w istniejące komponenty.
2. **Utrzymanie**: Modułowa architektura ułatwia zarządzanie kodem i jego utrzymanie, ponieważ zmiany w jednym komponencie nie wpływają bezpośrednio na inne, co jest kluczowe w długoterminowym rozwoju oprogramowania.

6.Interfejs użytkownika:

Interakcja z użytkownikiem odbywa się poprzez menu konsolowe, które umożliwia wybór pomiędzy wyświetleniem bieżącego czasu, ustawieniem alarmu, ustawieniem timera, oraz wyjściem z aplikacji. Do zarządzania wyświetlaniem i czyszczeniem ekranu wykorzystywane są funkcje takie jak `clearScreen()` oraz biblioteka `conio.h`.

7.Opis instalacji, uruchomienia i użytkowania:

1. Instalacja:
    - Skopiuj pliki źródłowe projektu do wybranego katalogu.
    - Otwórz terminal systemowy i przejdź do katalogu z programem
    - Skompiluj projekt przy użyciu kompilatora C++:
      `g++ main.cpp Clock.cpp Alarm.cpp Timer.cpp SevenSegmentDisplay.cpp -o zegarek`
    - Jeżeli korzystasz z Visual Studio, otwórz projekt jako solucję i zbuduj projekt używając zintegrowanego środowiska.

2. Uruchomienie:

   W terminalu, uruchom skompilowany plik wykonawczy:
   `./zegarek`

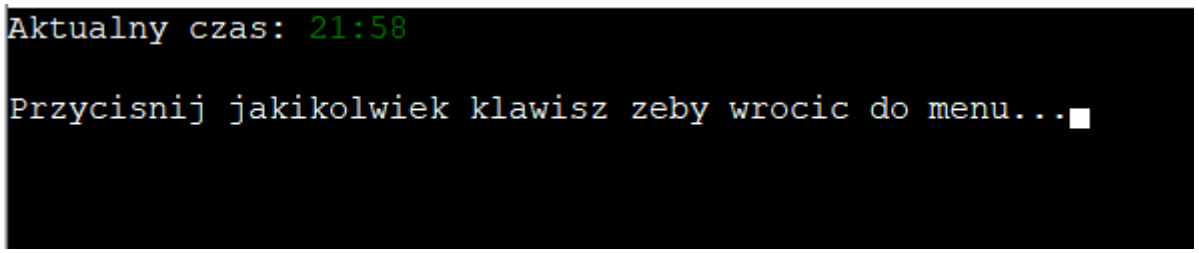
3. Użytkowanie:

   Ustawianie alarmu:
   `./zegarek -alarm 17:10`

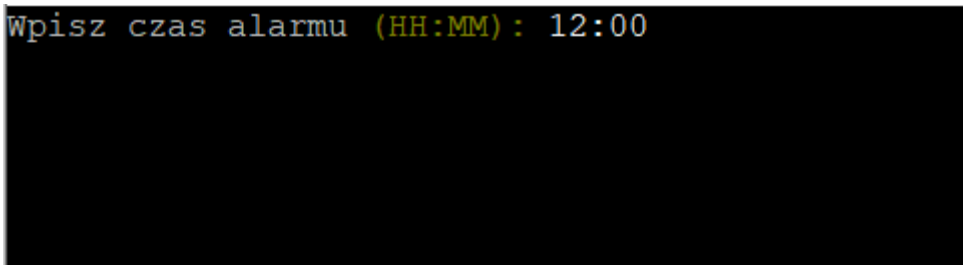
   Ustawienie timera:
   `./zegarek -timer 0:30`

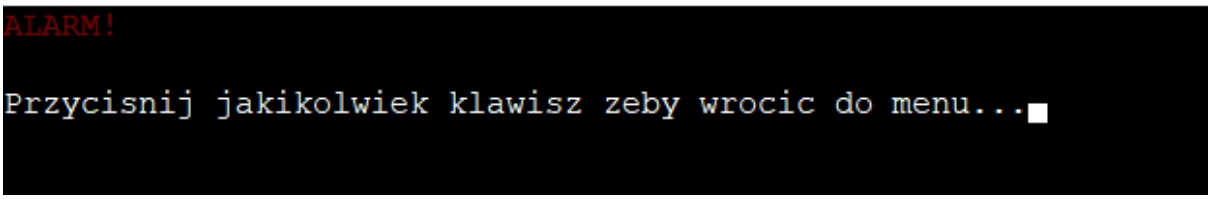
8.Szczegółowy opis funkcjonalności:

1. Pokazywanie bieżącego czasu:
    - Funkcja `showCurrentTime()` wykorzystuje metodę `render()` z klasy `Clock` do aktualizacji i wyświetlania aktualnego czasu w konsoli.

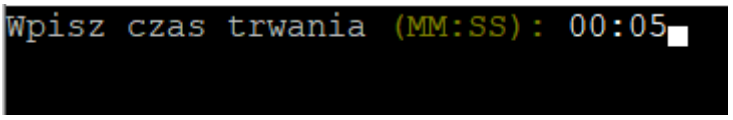


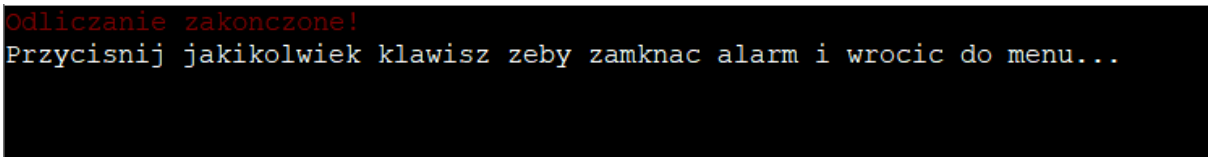
2. Ustawianie alarmu:
    - Użytkownik wpisuje żądany czas alarmu, który jest następnie przekazywany do metody `setAlarm()` klasy `Alarm`. System czeka na sygnał alarmu i odtwarza dźwięk, gdy alarm się uruchomi, co jest realizowane za pomocą funkcji `PlaySound()`.





3. Ustawianie timera:
    - Podobnie, użytkownik wpisuje czas trwania timera, który jest następnie przekazywany do metody `setTimer()` klasy `Timer`. Timer odlicza czas i aktualizuje wyświetlacz, a przy zakończeniu odliczania wydaje dźwięk zakończenia.





9.Wersja skompilowana i źródła:

- Dołączona do sprawozdania jest wersja skompilowana programu oraz kompletne źródła projektu z pełnym opisem wykonania i uruchomienia. Kody źródłowe zawierają odpowiednie komentarze i dokumentację pomagającą w zrozumieniu logiki programu oraz sposobu jego działania.

10.Dalsze możliwości rozwoju:

1. Dodanie graficznego interfejsu użytkownika (GUI) dla lepszej wizualizacji zegara i łatwiejszego zarządzania funkcjami.
2. Integracja z systemami mobilnymi za pomocą aplikacji mobilnej.
3. Rozszerzenie funkcjonalności o światowe strefy czasowe dla zegara.

11.Zagadnienia techniczne i wyzwania:

1. Synchronizacja wątków – zapewnienie, że aktualizacje czasu, alarmów i timerów działają równolegle bez zakłóceń.
2. Obsługa błędów związanych z nieprawidłowym wejściem od użytkownika oraz zarządzanie wyjątkami systemowymi.

12.Moduły i ich funkcje

a. Klasa `Clock`

- **Zadanie:** Zarządza wyświetlaniem aktualnego czasu.
- **Metody:**
  - `update()`: Metoda pusta, ponieważ aktualizacja czasu odbywa się dynamicznie podczas renderowania.
  - `render()`: Wywołuje metodę `getCurrentTime()`, a następnie przekazuje zwrócony czas do metody `display()` klasy `SevenSegmentDisplay` do wyświetlenia.
  - `getCurrentTime()`: Pobiera aktualny czas systemowy i formatuje go do postaci HH
    , używając funkcji C++ `strftime`.

```
1   #pragma once
2   #include <string>
3
4   class Clock {
5   public:
6       void update();
7       void render();
8
9   private:
10      std::string getCurrentTime();
11  };
12
```

b. Klasa `Alarm`

- **Zadanie:** Umożliwia ustawienie i zarządzanie alarmem.
- **Metody:**
  - `setAlarm(const std::string& time)`: Przypisuje czas, na który ma zostać ustawiony alarm.
  - `update()`: Sprawdza, czy obecny czas odpowiada czasowi alarmu i w razie potrzeby aktywuje alarm.
  - `render()`: Jeśli alarm dzwoni, wyświetla komunikat "ALARM!" za pomocą metody `display()` klasy `SevenSegmentDisplay`.
  - `isAlarmRinging()`: Zwraca stan alarmu, czyli czy dzwoni, czy nie.
  - `isAlarmTime()`: Prywatna metoda porównująca aktualny czas z ustawionym czasem alarmu.

```
1   #pragma once
2   #include <string>
3
4   class Alarm {
5   public:
6       void setAlarm(const std::string& time);
7       void update();
8       void render();
9       bool isAlarmRinging() const;
10
11  private:
12      std::string alarmTime;
13      bool alarmRinging = false;
14      bool isAlarmTime();
15  };
16
```

c. Klasa `Timer`

- **Zadanie:** Zarządza odliczaniem czasu.
- **Metody:**
  - `setTimer(const std::string& duration)`: Ustawia timer na podaną liczbę minut i sekund, przekształcając je na sekundy.
  - `update()`: Odlicza czas i aktualizuje wyświetlacz, zatrzymując timer po osiągnięciu zera.
  - `render()`: Wyświetla pozostały czas timera na ekranie, używając metody `display()` klasy `SevenSegmentDisplay`.
  - `formatTime(int seconds)`: Formatuje czas z sekund na format MM

```cpp
#pragma once
#include <string>

class Timer {
public:
    void setTimer(const std::string& duration);
    void update();
    void render();
    bool isRunning() const;

private:
    int remainingTime;
    bool running;
    std::string formatTime(int seconds);
};

```

d. Klasa `SevenSegmentDisplay`

- **Zadanie:** Symulacja wyświetlacza siedmiosegmentowego.
- **Metoda:**
    - `display(const std::string& text)`: Wyświetla przekazany tekst, imitując działanie wyświetlacza siedmiosegmentowego.

```cpp
#pragma once
#include <string>

class SevenSegmentDisplay {
public:
    static void display(const std::string& text);
};

```

13.Wykorzystanie i integracja klas

- **Integracja:** Każda z klas (`Clock`, `Timer`, `Alarm`) korzysta z klasy `SevenSegmentDisplay` do wyświetlania odpowiednich informacji na ekranie, co pokazuje modularność i reużywalność komponentów.
- **Zarządzanie stanem:** Aplikacja zarządza stanem różnych funkcji, takich jak bieżący czas, czas do alarmu, i czas do zakończenia timera, co pozwala na dynamiczne zarządzanie czasem w odpowiedzi na interakcję użytkownika.

14.Załączniki:

- main.cpp
- Clock.cpp, Clock.h
- Alarm.cpp, Alarm.h
- Timer.cpp, Timer.h
- SevenSegmentDisplay.cpp, SevenSegmentDisplay.h

Opis programu w języku Python.

Oto szczegółowo rozbudowana dokumentacja techniczna projektu zegarowego w języku Python.

1.Cele i założenia projektu:

Projekt ma na celu stworzenie aplikacji zegarowej, która oferuje funkcje zegarka, budzika i timera. Główne cele projektu to dokładne śledzenie czasu, umożliwienie użytkownikowi ustawiania alarmów oraz timery w formacie MM

2.Zastosowanie projektu:

Aplikacja może być wykorzystywana codziennie jako praktyczne narzędzie do pomiaru czasu i alarmowania o określonych wydarzeniach lub interwałach czasowych.

3.Technologie i oprogramowanie użyte w projekcie:

Projekt został napisany w języku Python 3.x, wykorzystując moduły standardowe takie jak `time`, `datetime`, `argparse`, `threading`, `sched`.

4.Wymagania do uruchomienia:

- Interpreter Python 3.x zainstalowany na urządzeniu.
- Brak konieczności instalacji dodatkowych zewnętrznych bibliotek.

5.Architektura aplikacji:

Aplikacja składa się z głównej pętli sterującej menu, która pozwala użytkownikowi wybrać opcje zegara, ustawienia alarmu lub timera. Obsługa czasu odbywa się w oddzielnych wątkach (`threading`), co umożliwia równoczesne wyświetlanie bieżącego czasu i obsługę alarmów/timerów. Do schedulowania alarmów wykorzystano moduł `sched`, który pozwala na ustawienie zadań do wykonania w przyszłości.

6.Opis diagramu klas:

W projekcie nie ma formalnego diagramu klas, ponieważ Python jako język skryptowy nie wymaga tak szczegółowych struktur. Funkcjonalności są zorganizowane w funkcje i wątki, które współpracują ze sobą bez formalnego modelu klas.

7.Implikacje dla rozwoju i utrzymania:

Projekt można łatwo rozwijać i rozbudowywać o dodatkowe funkcjonalności, takie jak obsługa wielu alarmów, integracja z interfejsem graficznym (GUI) lub zapisywanie ustawień między sesjami. Utrzymanie aplikacji wymaga dbałości o synchronizację wątków, obsługę wyjątków oraz zapewnienie dokładności pomiaru czasu.

8.Interfejs użytkownika:

Aplikacja oferuje tekstowy interfejs użytkownika, który umożliwia wybór opcji za pomocą klawiatury. Po uruchomieniu użytkownik może:

- Wyświetlać bieżący czas.

```
Menu:
1. Pokaż aktualny czas
2. Ustaw alarm
3. Wyjście
Wybierz opcję (1/2/3): 1
21:21:43 (naciśnij Enter aby wrócić do menu)
```

- Ustawiać alarmy na określone godziny i minuty.

```
Menu:
1. Pokaż aktualny czas
2. Ustaw alarm
3. Wyjście
Wybierz opcję (1/2/3): 2
Podaj godzinę alarmu (0-23): 21
Podaj minutę alarmu (0-59): 35
Czas do alarmu: 04:16 (naciśnij Enter aby wrócić do menu)
```

- Kończyć działanie programu

```
Menu:
1. Pokaż aktualny czas
2. Ustaw alarm
3. Wyjście
Wybierz opcję (1/2/3): 3
Zakończenie programu.

Process finished with exit code 0
```

9.Opis instalacji, uruchomienia i użytkowania:

a.Instalacja:

- Pobierz plik `project.py` z repozytorium.
- Upewnij się, że masz zainstalowany interpreter Python 3.x na swoim systemie.

b.Uruchomienie:

- Otwórz terminal lub wiersz poleceń.
- Przejdź do katalogu, w którym znajduje się plik `project.py`.
- Uruchom aplikację komendą `python project.py`.

c.Użytkowanie:

- Po uruchomieniu aplikacji postępuj zgodnie z opcjami wyświetlanymi w menu.
- Aby zakończyć działanie programu, wybierz odpowiednią opcję z menu.
-

10.Szczegółowy opis funkcjonalności:

1. Pokazywanie bieżącego czasu:
   - Funkcja `display_time(stop_event)` wypisuje aktualny czas co sekundę. Użytkownik może wcisnąć Enter, aby wrócić do menu głównego.
2. Ustawianie alarmu:
   - Funkcja `set_alarm(scheduler, alarm_hour, alarm_minute)` pozwala na ustawienie alarmu na określoną godzinę i minutę. Alarm zostanie uruchomiony w przyszłości, a czas do jego wyzwolenia jest wyświetlany na bieżąco. Po osiągnięciu czasu alarmu aplikacja informuje użytkownika o jego wystąpieniu.
3. Ustawianie timera:
   - Funkcja `set_timer(timer_duration)` pozwala na ustawienie timera na określony czas w formacie MM

. Po upływie tego czasu użytkownik otrzymuje powiadomienie o zakończeniu odliczania.

11.Dalsze możliwości rozwoju:

Projekt może być rozwijany poprzez dodanie następujących funkcji:

- Obsługa powtarzalnych alarmów.
- Integracja z systemowym zegarem i kalendarzem.
- Dodanie dźwięków alarmowych.
- Implementacja interfejsu graficznego (GUI).

12.Zagadnienia techniczne i wyzwania:

- Synchronizacja wątków: Zapewnienie, aby różne części aplikacji działały równocześnie i bez konfliktów.
- Obsługa wyjątków: Wychwytywanie błędów wprowadzanych przez użytkownika, takich jak nieprawidłowy format czasu.
- Precyzja odliczania czasu: Zapewnienie dokładności odliczania w timera i dokładności wyzwalania alarmów.

13.Moduły i ich funkcje:

Projekt wykorzystuje następujące funkcje zorganizowane w modułach:

- **Moduł główny (`project.py`)**: Zawiera funkcje obsługi menu, interakcji z użytkownikiem oraz główną pętlę programu.
- **Moduł `time`**: Wykorzystywany do zarządzania czasem rzeczywistym i pomiaru czasu.
- **Moduł `datetime`**: Służy do manipulacji datami i czasem, niezbędny do ustawiania alarmów.
- **Moduł `argparse`**: Umożliwia obsługę argumentów linii poleceń, co pozwala na uruchamianie programu z różnymi opcjami (np. ustawianie alarmów, timerów).
- **Moduł `threading`**: Wykorzystany do tworzenia i zarządzania wątkami, co umożliwia równoczesne wykonywanie różnych zadań (np. wyświetlanie bieżącego czasu, obsługa alarmów).
- **Moduł `sched`**: Służy do schedulowania zadań, takich jak wyzwalanie alarmów w określonym czasie.

14.Wykorzystanie i integracja klas:

Projekt nie korzysta z klas w tradycyjnym sensie (nie ma formalnych klas), ale funkcje są logicznie zorganizowane i współpracują ze sobą poprzez przekazywanie argumentów i korzystanie z modułów Pythona.

15.Załączniki:

Brak

Dokumentacja techniczna projektu zegarowego w Pythonie obejmuje wszystkie kluczowe elementy, od opisu celów i założeń po szczegółowe omówienie funkcji oraz technicznych aspektów realizacji.