C++ - Wydarzenia (obiektowe)

Ogólne

Program umieszcza w kalendarzu utworzone przez użytkownika wydarzenia. Każde wydarzenie to jeden obiekt oparty o klasę *Event*. Wprowadzone do kalendarza wydarzenia, można potem wyświetlić na ekranie. Cały projekt składa się z trzech plików (klasa i nagłówki metod są w osobnych plikach).

Uwaga: Nigdy nie wolno kopiować kodu z PDF-a, gdyż zawiera on niewidoczne znaki końca linii, twarde odstępy, odmienne cudzysłowy, odmienne apostrofy, odmienne myślniki. To godziny dodatkowej pracy na wykrywanie błędów i ich poprawianie. Kod należy przepisać ze zrozumieniem.

Kod - wersja podstawowa

Plik main.cpp

```
#include <iostream>
#include "event.h"

using namespace std;

int main() {
    Event wydarzenie[3]; // Trzy obiekty w tablicy
    cout << endl << "WPROWADZANIE DANYCH..." << endl;
    for (int i = 0; i < 3; i++) {
        wydarzenie[i].wprowadzWydarzenie();
    }

    cout << "\nNaciśnij ENTER, aby wyświetlić zawartość kalendarza...";
    cin.get();
    cout << "\nWprowadzone wydarzenia:";
    for (auto pojedynczeWydarzenie : wydarzenie) {
        pojedynczeWydarzenie.pokazWydarzenie();
    }

    cout << "\nNaciśnij ENTER, aby zakończyć..." << endl;
    cin.get();
    return 0;
}
```

Plik event.h

```
#include <iostream>

using namespace std;
```

```
class Event {
   private:
      int dzien, miesiac, rok, godzina, minuta;
      string nazwaWydarzenia;
   public:
      // Konstruktor:
      Event(string nazWyd="Brak wydarzenia", int dd=1, int mm=1, int yyyy=1970, int hh=00,
      int mins=00);

      // Destruktor:
      ~Event();

      // Deklarujemy metody:
      void wprowadzWydarzenie();
      void pokazWydarzenie();
};
```

Plik event.cpp

```
#include <iostream>
#include "event.h"

using namespace std;
string liczbaJakoString;

// ================ METODA: WPROWADŹ WYDARZENIE ==============//
void Event::wprowadzWydarzenie() {
   cout << "Nazwa wydarzenia: ";
   getline(cin, nazwaWydarzenia);

   cout << "Dzień: ";
   getline(cin, liczbaJakoString);
   dzien = stoi(liczbaJakoString);

   cout << "Miesiąc: ";
   getline(cin, liczbaJakoString);
   miesiac = stoi(liczbaJakoString);

   cout << "Rok: ";
   getline(cin, liczbaJakoString);
   rok = stoi(liczbaJakoString);

   cout << "Godzina: ";
   getline(cin, liczbaJakoString);
   godzina = stoi(liczbaJakoString);

   cout << "Minuta: ";
   getline(cin, liczbaJakoString);
   minuta = stoi(liczbaJakoString);
```

```
    cout << endl;
}

// ================== METODA: POKAŻ WYDARZENIE ============== //
void Event::pokazWydarzenie() {
    cout << endl << endl << nazwaWydarzenia << endl << dzien << "." << miesiac << "."
    << rok << endl << godzina << ":" << minuta << endl;
}

// ================== KONSTRUKTOR Z PARAMETRAMI ================ //
Event::Event(string nazWyd, int dd, int mm, int yyyy, int hh, int mins) {
    nazwaWydarzenia = nazWyd;
    dzien = dd;
    miesiac = mm;
    rok = yyyy;
    godzina = hh;
    minuta = mins;
}

// ==================== DESTRUKTOR ==================== //
Event::~Event() {
    // cout << "\nZostał wywołany destruktor." << endl;
}
```

Kod - wersja rozbudowana (*Windows*)

// Projekt składa się z trzech plików: głównego ***main.cpp*** oraz dwóch innych, które należy
// włączyć do projektu: ***event.h*** i ***event.cpp***.

Plik main.cpp

```
#include <iostream>
#include <algorithm> // Potrzebne do find()
#include "event.h"
#include <vector>
#include <windows.h> // Potrzebne do aktywacji kolorów w konsoli

using namespace std;

int main() {
    // ================== KOLOROWANIE NAPISÓW ================
    // Tworzymy tzw. uchwyt do tego, co będzie pojawiać się na konsoli (do bufora konsoli):
    HANDLE konsola = GetStdHandle(STD_OUTPUT_HANDLE);
    // Aktywujemy virtualny terminal i to, co będzie się na nim pojawiać:
    #ifndef ENABLE_VIRTUAL_TERMINAL_PROCESSING // Jeśli nie jest zdefiniowany, to:
    #define ENABLE_VIRTUAL_TERMINAL_PROCESSING 0x0004
    #endif
    // Aby powyższe działało, musimy aktywować i to, co poniżej:
```

```
#ifndef ENABLE_PROCESSED_OUTPUT // Jeśli nie jest zdefiniowany, to:
#define ENABLE_PROCESSED_OUTPUT 0x0001
#endif

// Wartość trybu (intup lub output). Słowo "dw" to skrót od "Display Window",
// jest to jednak nazwa zmiennej, i może być inna:
DWORD dwMode = 0;
dwMode |= ENABLE_PROCESSED_OUTPUT | ENABLE_VIRTUAL_TERMINAL_PROCESSING;
SetConsoleMode(konsola, dwMode);
// ============== KOLOROWANIE NAPISÓW - KONIEC ==============

string  tablicaOdpowiedziTak[]  = {"Tak", "tak", "T", "t", "Yes", "yes", "y", "Y", "taa", "yee",
"ta", "Ta"};
string odpowiedzCin, odpowiedzCout;
bool jestZgodaCin = true; // W C++ nie ma domyślnych wartości boolowskich, dlatego
// przypisujemy tutaj
// Nie wiemy z góry, ile obiektów postanowi stworzyć użytkownik, więc nie możemy użyć
// tablicy obiektów (ona bowiem wymaga podania rozmiaru tablicy jeszcze przed kompilacją
// programu). Zatem, zamiast określonej rozmiarowo tablicy na obiekty, tworzymy sobie
// kontener na obiekty z typem danych określonym w klasie Event:
vector<Event> kontenerNaWydarzenia;

// Tworzymy obiekt w oparciu o klasę Event. Ale obiekt już w momencie powstania powinien
// mieć wartości początkowe.
// Dlatego zazwyczaj wywołujemy konstruktor, aby przekazać mu argumenty, np.:
// Event wydarzenie1("Likwidacja Warszawy", 15, 11, 2025, 8, 20);
// Chyba że wartości domyślne są już utworzone w definicji konstruktora (tzw. konstruktor
// przypisujący), w takim przypadku możemy po prostu utworzyć obiekt bez argumentów:
Event wydarzenie;

cout << "\n\033[1;36;40m===== KALENDARZ: WYDARZENIA =====\033[0m" << endl;
cout << endl << "WPROWADZANIE DANYCH..." << endl;
while (jestZgodaCin) {
    // Wprowadzamy wydarzenie do kalendarza:
    wydarzenie.wprowadzWydarzenie();
    // Poniżej, umieszczamy obiekt "wydarzenie" w kontenerze. W momencie dodawania
    // obiektów do kontenera, są one automatycznie indeksowane (umieszczane w osobnych
    // indeksach), np. kontenerNaWydarzenia[1], kontenerNaWydarzenia[2], itd.
    // Dzięki temu obiekty, które mają tę samą nazwę "wydarzenie" są w kontenerze
    // odróżnialne, bo mają różne indeksy:
    kontenerNaWydarzenia.push_back(wydarzenie);

    cout << "Czy wprowadzić następne wydarzenie? (t/n): ";
    getline(cin, odpowiedzCin);
    // Poniżej, find() porównuje każdy z elementów tablicy do szukanej wartości i zwraca "1"
    // lub "0".
    // begin() = pierwszy element pierwszej przeszukiwanej tablicy;
    // end() = ostatni element ostatniej przeszukiwanej tablicy (tu jest akurat tą samą).
```

```
    // Jeśli znajdzie, zatrzymuje się (znaleziony element nie jest tożsamy z ostatnim, więc
    // całość jest prawdą). Jeśli nie znajdzie i dojdzie do końca, wskaże na ostatni element
    // (zwróci go), a ostatni sprawdzany element będzie tożsamy z ostatnim elementem
    // tablicy, co znaczy, że nie znalazł (całość więc zwróci 0).
    // find() - jeśli jest w stanie bezproblemowo przeszukać wszystkie elementy tablicy -
    // zawsze zwraca 1 bez względu na to, czy znajdzie czy nie (to jest tylko informacja, że
    // udało się przeszukanie). Dopiero porównanie z ostatnim elementem w tablicy daje
    // odpowiedź, czy znaleziono.
    JestZgodaCin = find(begin(tablicaOdpowiedziTak), end(tablicaOdpowiedziTak),
    odpowiedzCin) != end(tablicaOdpowiedziTak);
  }

  cout << "\nCzy chcesz wyświetlić zawartość kalendarza? (t/n): ";
  getline(cin, odpowiedzCout);
  bool jestZgodaCout = find(begin(tablicaOdpowiedziTak), end(tablicaOdpowiedziTak),
  odpowiedzCout) != end(tablicaOdpowiedziTak);
  if (jestZgodaCout) {
    cout << "\n\033[1;30;40mWprowadzone wydarzenia:\033[0m";
    // Wyrażanie "auto" oznacza automatyczne wykrywanie typu danych kolejnych obiektów
    // znajdujących się w kontenerze.
    // Poniżej, pętla zakresowa (wprowadzona w C++11), w której następuje iteracja po
    // każdym elemencie kontenera (nie musimy określać licznika, ilość elementów w
    // kontenerze zostanie automatycznie wykryta).
    // W poniższej pętli możemy utworzyć zmienną o nazwie "pojedynczeWydarzenie", ale
    // wtedy tworzymy w pamięci kopię obiektu poprzez przypisanie go do tej zmiennej
    // (zmienna jest wtedy kopią obiektu). Lepiej więc będzie użyć referencji
    // "&pojedynczeWydarzenie", która przy każdym kolejnym przejściu wskazywać będzie na
    // oryginał każdego obiektu znajdującego się w kontenerze (na jego indeks).
    // Referencja jest tutaj wyrażeniem o zasięgu miejscowym i działa tylko wewnętrz pętli
    // FOR.
    for (auto &pojedynczeWydarzenie : kontenerNaWydarzenia) {
      pojedynczeWydarzenie.pokazWydarzenie();
    }
  }

  cout << "\n\033[1;30;40mNaciśnij ENTER, aby zakończyć...\033[0m" << endl;
  cin.get();
  return 0;
}
```

Plik event.h

```
#include <iostream>

using namespace std;

class Event {
  // Dokonujemy tzw. hermetyzacji danych (umieszczamy je w "szczelnym słoiku, czyli w sekcji
  // prywatnej"):
```

```
    private:
        int dzien, miesiac, rok, godzina, minuta;
        string nazwaWydarzenia;
    public:
        // Tworząc kontruktor wystarczy tylko wpisać typ danych: Event(string, int, int, int, int, int);
        // Poniżej jednak utworzymy "konstruktor przypisujący" (domyślne wartości):
        Event(string nazWyd="Brak wydarzenia", int dd=1, int mm=1, int yyyy=1970, int hh=00,
        int mins=00);
        // Jeśli konstruktor nie zostanie zdefiniowany, to i tak zostanie utworzony "konstruktor
        // domiemany".

        // Destruktor:
        ~Event();

        // Deklarujemy metody:
        bool czyToJestLiczba(string liczbaJakoString);
        void wprowadzWydarzenie();
        void pokazWydarzenie();
};
```

Plik event.cpp

```
#include <iostream>
#include "event.h"

using namespace std;

string liczbaJakoString;
int dlugoscWpisu  = liczbaJakoString.length();

// ================= METODA: CZY TO JEST LICZBA? ==============//
bool Event::czyToJestLiczba(string liczbaJakoString) {
    for (int licznik = 0; licznik < dlugoscWpisu; licznik++) {
        if (isdigit(liczbaJakoString[licznik]) == false) {
            return false;
        }
    }
    return true;
}


// ============== METODA: WPROWADŹ WYDARZENIE ============//
void Event::wprowadzWydarzenie() {
    bool wprowadzLiczbeDzien = true; // W C++ zmienne logiczne nie mają domyślnej wartości
    bool wprowadzLiczbeMiesiac = true;
    bool wprowadzLiczbeRok = true;
    bool wprowadzLiczbeGodzina = true;
    bool wprowadzLiczbeMinuta = true;
    cout << "Nazwa wydarzenia: ";
```

```
// Nazwa wydarzenia może zawierać wiele wyrażeń, dlatego wczytujemy całą linię:
getline(cin, nazwaWydarzenia);

// Skoro już raz użyliśmy getline, to używamy go wszędzie poniżej.
// Wprowadzamy dzień:
while (wprowadzLiczbeDzien) {
   cout << "Dzień: ";
   getline(cin, liczbaJakoString);
   if (czyToJestLiczba(liczbaJakoString) == false) {
      cout << "\033[1;31;40mTo nie jest liczba naturalna mniejsza od 31! Spróbuj jeszcze
      raz...\033[0m\n" << endl;
   }
    else {
      if (stoi(liczbaJakoString) <= 31) {
         dzien = stoi(liczbaJakoString);
         wprowadzLiczbeDzien = false; // Jeśli wprowadzono liczbę, wyłącz pętlę WHILE
      }
       else {
         cout << "\033[1;31;40mDzień nie może być większy niż 31!\033[0m\n" << endl;
      }
   }
}

// Wprowadzamy miesiąc:
while (wprowadzLiczbeMiesiac) {
   cout << "Miesiąc: ";
   getline(cin, liczbaJakoString);
   if (czyToJestLiczba(liczbaJakoString) == false) {
      cout << "\033[1;31;40mTo nie jest liczba naturalna mniejsza od 12! Spróbuj jeszcze
      raz...\033[0m\n" << endl;
   }
    else {
      if (stoi(liczbaJakoString) <= 12) {
         miesiac = stoi(liczbaJakoString);
         wprowadzLiczbeMiesiac = false; // Jeśli wprowadzono liczbę, wyłącz pętlę WHILE
      }
       else {
         cout << "\033[1;31;40mMiesiąc nie może być większy niż 12!\033[0m\n" << endl;
      }
    }
}

// Wprowadzamy rok:
while (wprowadzLiczbeRok) {
   cout << "Rok: ";
   getline(cin, liczbaJakoString);
   if (czyToJestLiczba(liczbaJakoString) == false) {
      cout << "\033[1;31;40mTo nie jest liczba naturalna z zakresu 2024-2150! Spróbuj
```

```
        jeszcze raz...\033[0m\n" << endl;
    }
    else {
        if ((stoi(liczbaJakoString) >= 2024) && (stoi(liczbaJakoString) <= 2150)) {
            rok = stoi(liczbaJakoString);
            wprowadzLiczbeRok = false; // Jeśli wprowadzono liczbę, wyłącz pętlę WHILE
        }
        else {
            cout << "\033[1;31;40mRok musi mieścić się w zakresie 2024-2150!\033[0m\n" <<
            endl;
        }
    }
}

// Wprowadzamy godzinę:
while (wprowadzLiczbeGodzina) {
    cout << "Godzina: ";
    getline(cin, liczbaJakoString);
    if (czyToJestLiczba(liczbaJakoString) == false) {
        cout << "\033[1;31;40mTo nie jest liczba naturalna z zakresu 0-24! Spróbuj jeszcze
        raz...\033[0m\n" << endl;
    }
    else {
        if (stoi(liczbaJakoString) <= 24) {
            godzina = stoi(liczbaJakoString);
            wprowadzLiczbeGodzina = false; // Jeśli wprowadzono liczbę, wyłącz pętlę WHILE
        }
        else {
            cout << "\033[1;31;40mGodzina nie może być większa niż 24!\033[0m\n" << endl;
        }
    }
}

// Wprowadzamy minutę:
while (wprowadzLiczbeMinuta) {
    cout << "Minuta: ";
    getline(cin, liczbaJakoString);
    if (czyToJestLiczba(liczbaJakoString) == false) {
        cout << "\033[1;31;40mTo nie jest liczba naturalna z zakresu 0-59! Spróbuj jeszcze
        raz...\033[0m\n" << endl;
    }
    else {
        if (stoi(liczbaJakoString) <= 59) {
            minuta = stoi(liczbaJakoString);
            wprowadzLiczbeMinuta = false; // Jeśli wprowadzono liczbę, wyłącz pętlę WHILE
        }
        else {
            cout << "\033[1;31;40mMinuta nie może być większa niż 59!\033[0m\n" << endl;
```

```
        }
      }
    }
    cout << endl;
}



// ================ METODA: POKAŻ WYDARZENIE ============== //
void Event::pokazWydarzenie() {
    cout << endl << "\033[1;33;40m" << endl << nazwaWydarzenia << endl << dzien << "."
    << miesiac << "." << rok << endl << godzina << ":" << minuta << "\033[0m" << endl;
}



// ================ KONSTRUKTOR Z PARAMETRAMI ================ //
Event::Event(string nazWyd, int dd, int mm, int yyyy, int hh, int mins) {
    // Przypisanie podanych argumentów do atrybutów prywatnych:
    nazwaWydarzenia = nazWyd;
    dzien = dd;
    miesiac = mm;
    rok = yyyy;
    godzina = hh;
    minuta = mins;
}



// ==================== DESTRUKTOR ================== //
Event::~Event() {
    // cout << "\nZostał wywołany destruktor." << endl;
}
```

Kod - wersja rozbudowana (*Linux*)

Projekt składa się z trzech plików: głównego ***main.cpp*** oraz dwóch innych, które należy włączyć do projektu: ***event.h*** i ***event.cpp***.

Plik main.cpp

```
#include <iostream>
#include <algorithm> // Potrzebne do find()
#include "event.h"
#include <vector>

using namespace std;

int main() {
    string  tablicaOdpowiedziTak[]  = {"Tak", "tak", "T", "t", "Yes", "yes", "y", "Y", "taa", "yee",
```

```
"ta", "Ta"};
string odpowiedzCin, odpowiedzCout;
bool jestZgodaCin = true; // W C++ nie ma domyślnych wartości boolowskich, dlatego
// przypisujemy tutaj
// Nie wiemy z góry, ile obiektów postanowi stworzyć użytkownik, więc nie możemy użyć
// tablicy obiektów (ona bowiem wymaga podania rozmiaru tablicy jeszcze przed kompilacją
// programu). Zatem, zamiast określonej rozmiarowo tablicy na obiekty, tworzymy sobie
// kontener na obiekty z typem danych określonym w klasie Event:
vector<Event> kontenerNaWydarzenia;

// Tworzymy obiekt w oparciu o klasę Event. Ale obiekt już w momencie powstania powinien
// mieć wartości początkowe.
// Dlatego zazwyczaj wywołujemy konstruktor, aby przekazać mu argumenty, np.:
// Event wydarzenie1("Likwidacja Warszawy", 15, 11, 2025, 8, 20);
// Chyba że wartości domyślne są już utworzone w definicji konstruktora (tzw. konstruktor
// przypisujący), w takim przypadku możemy po prostu utworzyć obiekt bez argumentów:
Event wydarzenie;

cout << "\n\033[1;36;40m===== KALENDARZ: WYDARZENIA =====\033[0m" << endl;
cout << endl << "WPROWADZANIE DANYCH..." << endl;
while (jestZgodaCin) {
    // Wprowadzamy wydarzenie do kalendarza:
    wydarzenie.wprowadzWydarzenie();
    // Poniżej, umieszczamy obiekt "wydarzenie" w kontenerze. W momencie dodawania
    // obiektów do kontenera, są one automatycznie indeksowane (umieszczane w osobnych
    // indeksach), np. kontenerNaWydarzenia[1], kontenerNaWydarzenia[2], itd.
    // Dzięki temu obiekty, które mają tę samą nazwę "wydarzenie" są w kontenerze
    // odróżnialne, bo mają różne indeksy:
    kontenerNaWydarzenia.push_back(wydarzenie);

    cout << "Czy wprowadzić następne wydarzenie? (t/n): ";
    getline(cin, odpowiedzCin);
    // Poniżej, find() porównuje każdy z elementów tablicy do szukanej wartości i zwraca "1"
    // lub "0".
    // begin() = pierwszy element pierwszej przeszukiwanej tablicy;
    // end() = ostatni element ostatniej przeszukiwanej tablicy (tu jest akurat tą samą).
    // Jeśli znajdzie, zatrzymuje się (znaleziony element nie jest tożsamy z ostatnim, więc
    // całość jest prawdą). Jeśli nie znajdzie i dojdzie do końca, wskaże na ostatni element
    // (zwróci go), a ostatni sprawdzany element będzie tożsamy z ostatnim elementem
    // tablicy, co znaczy, że nie znalazł (całość więc zwróci 0).
    // find() - jeśli jest w stanie bezproblemowo przeszukać wszystkie elementy tablicy -
    // zawsze zwraca 1 bez względu na to, czy znajdzie czy nie (to jest tylko informacja, że
    // udało się przeszukanie). Dopiero porównanie z ostatnim elementem w tablicy daje
    // odpowiedź, czy znaleziono.
    JestZgodaCin = find(begin(tablicaOdpowiedziTak), end(tablicaOdpowiedziTak),
    odpowiedzCin) != end(tablicaOdpowiedziTak);
}
```

```
    cout << "\nCzy chcesz wyświetlić zawartość kalendarza? (t/n): ";
    getline(cin, odpowiedzCout);
    bool jestZgodaCout = find(begin(tablicaOdpowiedziTak), end(tablicaOdpowiedziTak),
    odpowiedzCout) != end(tablicaOdpowiedziTak);
    if (jestZgodaCout) {
        cout << "\n\033[1;30;40mWprowadzone wydarzenia:\033[0m";
        // Wyrażanie "auto" oznacza automatyczne wykrywanie typu danych kolejnych obiektów
        // znajdujących się w kontenerze.
        // Poniżej, pętla zakresowa (wprowadzona w C++11), w której następuje iteracja po
        // każdym elemencie kontenera (nie musimy określać licznika, ilość elementów w
        // kontenerze zostanie automatycznie wykryta).
        // W poniższej pętli możemy utworzyć zmienną o nazwie "pojedynczeWydarzenie", ale
        // wtedy tworzymy w pamięci kopię obiektu poprzez przypisanie go do tej zmiennej
        // (zmienna jest wtedy kopią obiektu). Lepiej więc będzie użyć referencji
        // "&pojedynczeWydarzenie", która przy każdym kolejnym przejściu wskazywać będzie na
        // oryginał każdego obiektu znajdującego się w kontenerze (na jego indeks).
        // Referencja jest tutaj wyrażeniem o zasięgu miejscowym i działa tylko wewnętrz pętli
        // FOR.
        for (auto &pojedynczeWydarzenie : kontenerNaWydarzenia) {
            pojedynczeWydarzenie.pokazWydarzenie();
        }
    }

    cout << "\n\033[1;30;40mNaciśnij ENTER, aby zakończyć...\033[0m" << endl;
    cin.get();
    return 0;
}
```

Plik event.h

```
#include <iostream>

using namespace std;

class Event {
    // Dokonujemy tzw. hermetyzacji danych (umieszczamy je w "szczelnym słoiku, czyli w sekcji
    // prywatnej"):
    private:
        int dzien, miesiac, rok, godzina, minuta;
        string nazwaWydarzenia;
    public:
        // Tworząc kontruktor wystarczy tylko wpisać typ danych: Event(string, int, int, int, int, int);
        // Poniżej jednak utworzymy "konstruktor przypisujący" (domyślne wartości):
        Event(string nazWyd="Brak wydarzenia", int dd=1, int mm=1, int yyyy=1970, int hh=00,
        int mins=00);
        // Jeśli konstruktor nie zostanie zdefiniowany, to i tak zostanie utworzony "konstruktor
        // domiemany".

        // Destruktor:
```

```
    ~Event();

    // Deklarujemy metody:
    bool czyToJestLiczba(string liczbaJakoString);
    void wprowadzWydarzenie();
    void pokazWydarzenie();
};
```

Plik event.cpp

```
#include <iostream>
#include "event.h"

using namespace std;

string liczbaJakoString;
int dlugoscWpisu = liczbaJakoString.length();

// ================ METODA: CZY TO JEST LICZBA? ==============//
bool Event::czyToJestLiczba(string liczbaJakoString) {
   for (int licznik = 0; licznik < dlugoscWpisu; licznik++) {
      if (isdigit(liczbaJakoString[licznik]) == false) {
         return false;
      }
   }
   return true;
}


// =============== METODA: WPROWADŹ WYDARZENIE ==============//
void Event::wprowadzWydarzenie() {
   bool wprowadzLiczbeDzien = true; // W C++ zmienne logiczne nie mają domyślnej wartości
   bool wprowadzLiczbeMiesiac = true;
   bool wprowadzLiczbeRok = true;
   bool wprowadzLiczbeGodzina = true;
   bool wprowadzLiczbeMinuta = true;
   cout << "Nazwa wydarzenia: ";
   // Nazwa wydarzenia może zawierać wiele wyrażeń, dlatego wczytujemy całą linię:
   getline(cin, nazwaWydarzenia);

   // Skoro już raz użyliśmy getline, to używamy go wszędzie poniżej.
   // Wprowadzamy dzień:
   while (wprowadzLiczbeDzien) {
      cout << "Dzień: ";
      getline(cin, liczbaJakoString);
      if (czyToJestLiczba(liczbaJakoString) == false) {
         cout << "\033[1;31;40mTo nie jest liczba naturalna mniejsza od 31! Spróbuj jeszcze
         raz...\033[0m\n" << endl;
      }
```

```
    else {
      if (stoi(liczbaJakoString) <= 31) {
         dzien = stoi(liczbaJakoString);
         wprowadzLiczbeDzien = false; // Jeśli wprowadzono liczbę, wyłącz pętlę WHILE
      }
       else {
         cout << "\033[1;31;40mDzień nie może być większy niż 31!\033[0m\n" << endl;
      }
    }
}

// Wprowadzamy miesiąc:
while (wprowadzLiczbeMiesiac) {
   cout << "Miesiąc: ";
   getline(cin, liczbaJakoString);
   if (czyToJestLiczba(liczbaJakoString) == false) {
      cout << "\033[1;31;40mTo nie jest liczba naturalna mniejsza od 12! Spróbuj jeszcze
      raz...\033[0m\n" << endl;
   }
    else {
      if (stoi(liczbaJakoString) <= 12) {
         miesiac = stoi(liczbaJakoString);
         wprowadzLiczbeMiesiac = false; // Jeśli wprowadzono liczbę, wyłącz pętlę WHILE
      }
       else {
         cout << "\033[1;31;40mMiesiąc nie może być większy niż 12!\033[0m\n" << endl;
      }
    }
}

// Wprowadzamy rok:
while (wprowadzLiczbeRok) {
   cout << "Rok: ";
   getline(cin, liczbaJakoString);
   if (czyToJestLiczba(liczbaJakoString) == false) {
      cout << "\033[1;31;40mTo nie jest liczba naturalna z zakresu 2024-2150! Spróbuj
      jeszcze raz...\033[0m\n" << endl;
   }
    else {
      if ((stoi(liczbaJakoString) >= 2024) && (stoi(liczbaJakoString) <= 2150)) {
         rok = stoi(liczbaJakoString);
         wprowadzLiczbeRok = false; // Jeśli wprowadzono liczbę, wyłącz pętlę WHILE
      }
       else {
         cout << "\033[1;31;40mRok musi mieścić się w zakresie 2024-2150!\033[0m\n" <<
         endl;
      }
    }
```

```
    }

    // Wprowadzamy godzinę:
    while (wprowadzLiczbeGodzina) {
        cout << "Godzina: ";
        getline(cin, liczbaJakoString);
        if (czyToJestLiczba(liczbaJakoString) == false) {
            cout << "\033[1;31;40mTo nie jest liczba naturalna z zakresu 0-24! Spróbuj jeszcze
            raz...\033[0m\n" << endl;
        }
        else {
            if (stoi(liczbaJakoString) <= 24) {
                godzina = stoi(liczbaJakoString);
                wprowadzLiczbeGodzina = false; // Jeśli wprowadzono liczbę, wyłącz pętlę WHILE
            }
            else {
                cout << "\033[1;31;40mGodzina nie może być większa niż 24!\033[0m\n" << endl;
            }
        }
    }

    // Wprowadzamy minutę:
    while (wprowadzLiczbeMinuta) {
        cout << "Minuta: ";
        getline(cin, liczbaJakoString);
        if (czyToJestLiczba(liczbaJakoString) == false) {
            cout << "\033[1;31;40mTo nie jest liczba naturalna z zakresu 0-59! Spróbuj jeszcze
            raz...\033[0m\n" << endl;
        }
        else {
            if (stoi(liczbaJakoString) <= 59) {
                minuta = stoi(liczbaJakoString);
                wprowadzLiczbeMinuta = false; // Jeśli wprowadzono liczbę, wyłącz pętlę WHILE
            }
            else {
                cout << "\033[1;31;40mMinuta nie może być większa niż 59!\033[0m\n" << endl;
            }
        }
    }
    cout << endl;
}



// ================ METODA: POKAŻ WYDARZENIE ============== //
void Event::pokazWydarzenie() {
    cout << endl << "\033[1;33;40m" << endl << nazwaWydarzenia << endl << dzien << "."
    << miesiac << "." << rok << endl << godzina << ":" << minuta << "\033[0m" << endl;
```

```
}


// ================ KONSTRUKTOR Z PARAMETRAMI ================ //
Event::Event(string nazWyd, int dd, int mm, int yyyy, int hh, int mins) {
    // Przypisanie podanych argumentów do atrybutów prywatnych:
    nazwaWydarzenia = nazWyd;
    dzien = dd;
    miesiac = mm;
    rok = yyyy;
    godzina = hh;
    minuta = mins;
}


// ====================== DESTRUKTOR ================== //
Event::~Event() {
    // cout << "\nZostał wywołany destruktor." << endl;
}
```

Ostatnia aktualizacja: 25 stycznia 2024.