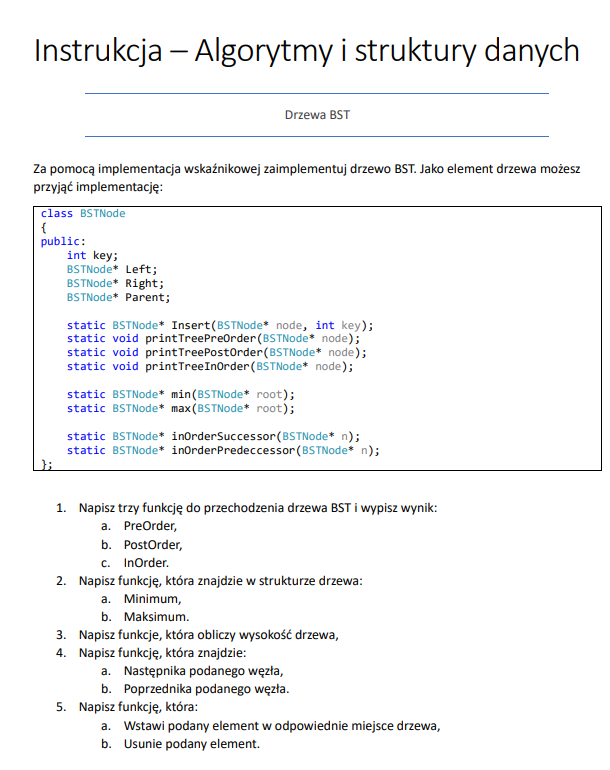
# Sprawozdanie, Zajęcia nr 2 – Drzewa BST.

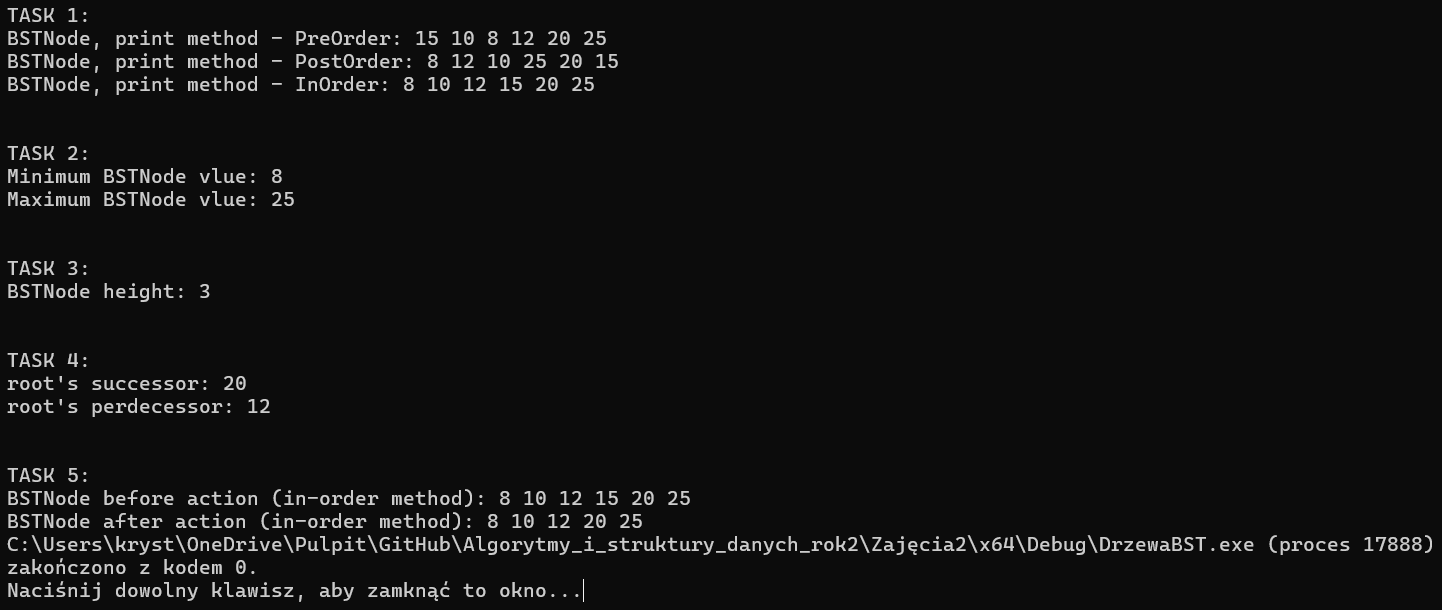
## Krystian Kostrzewa, 418845. WIMiIP, Informatyka Techniczna, sn. Data zajęć: 07.04.2024

1. **Drewa BST - wprowadzenie**

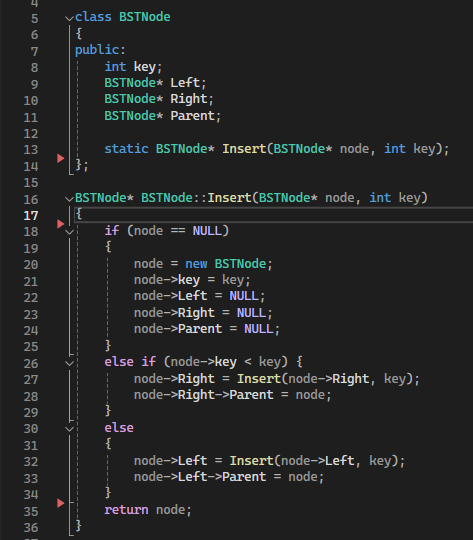
**Polecenia:**



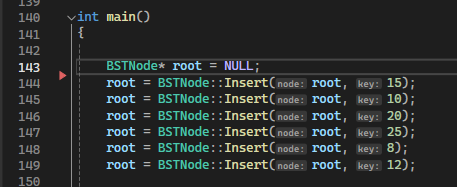
**Wynik w konsoli:**



**Kod i opis rozwiązania:**

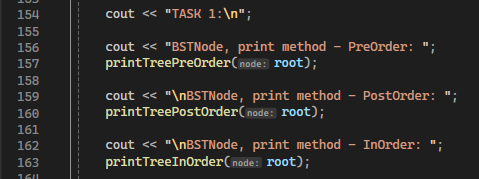


Cały kod działa w oparciu o strukturę *BSTNode*, która przedstawiona jest na powyższym screenie. Zmienna *key* przechowuje wartość liczbową danego elementu drzewa. Zmienne *Left, Right* i *Parent* służą do nawigacji po drzewie. Oznaczają one kolejno: lewy element pod elementem, prawy element pod elementem i przodka tego elementu. Funkcja *Insert* służy do dodawania nowych elementów do drzewa.

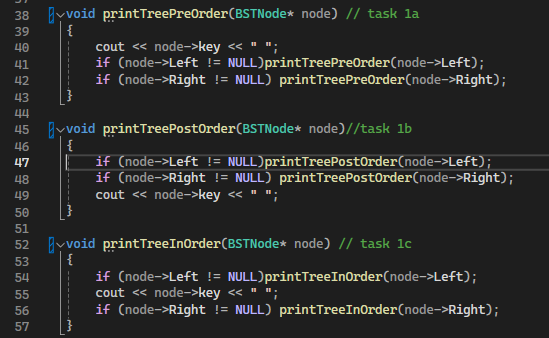


W funkcji main program rozpoczyna się od implementacji drzewa *root*, i uzupełnienia go pewnymi wartościami.

Rozwiązanie zadania 1:

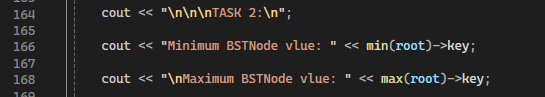


W funkcji main odbywa się proste wywołanie funkcji i opis, jaką metodą nastąpi wyświetlenie drzewa.

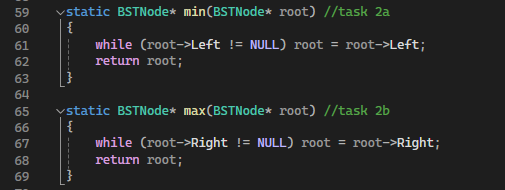


W powyższym fragmencie kodu przedstawione są funkcje wykorzystane w zadaniu 1. Wykorzystują rekurencyjne podejście, przechodząc po całym drzewie w odpowiedniej kolejności i wyświetlają wartości elementów drzewa.

Rozwiązanie zadania 2:



W funkcji main odbywa się proste wywołanie funkcji i opis co zostanie wyświetlone.

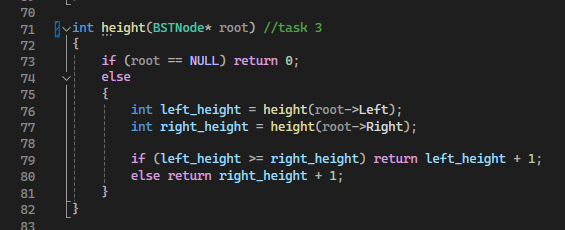


Funkcje działają na podobnej zasadzie. Na przykładzie *min* przedstawię ich zasadę działania. Od korzenia cały czas sprawdzany jest element po lewej stronie(mniejszy). Podążając cały czas taką ścieżką, w momencie kiedy kolejny element po lewej nie istnieje – wiadomo, że aktualny element jest najmniejszym w drzewie.

Rozwiązanie zadania 3:

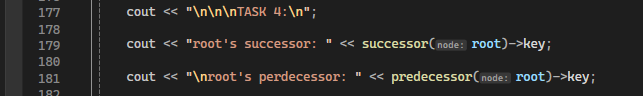


W funkcji main odbywa się proste wywołanie funkcji i opis co zostanie wyświetlone.

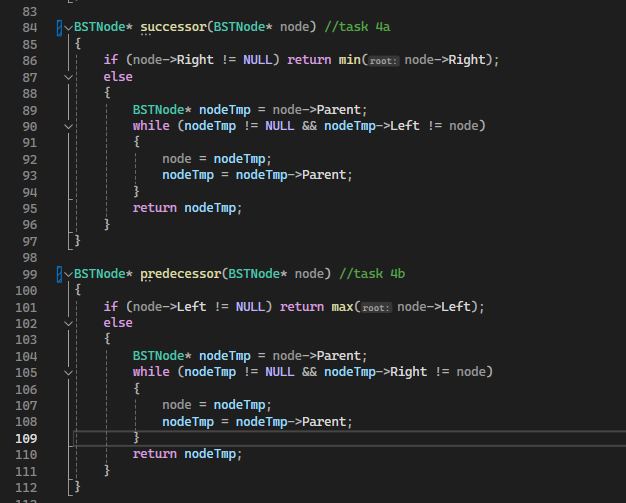


Działa ona na zasadzie rekurencyjnego podejścia. Sprawdza długość każdego węzła i zwraca długość najdłuższego z nich.

Rozwiązanie zadania 4:

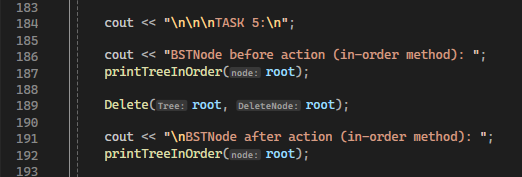


W funkcji main odbywa się proste wywołanie funkcji i opis co zostanie wyświetlone.

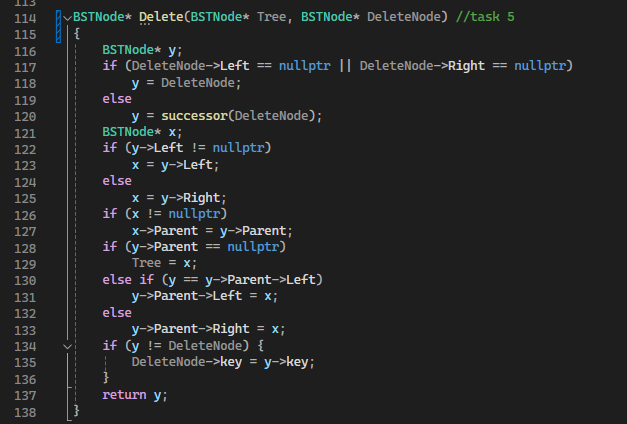


Funkcje działają na podobnej zasadzie. Na przykładzie *successor* przedstawię ich zasadę działania. Aby uzyskać najmniejszą większą wartość na początku program sprawdza, czy istnieje prawy element. Jeśli tak, zwraca najmniejszą wartość z tego poddrzewa. W przeciwnym razie, przechodzimy w górę po drzewie, aż znajdziemy węzeł, którego lewe dziecko jest węzłem początkowym ścieżki. Ten węzeł jest następnikiem i jest zwracany w funkcji.

Rozwiązanie zadania 5:



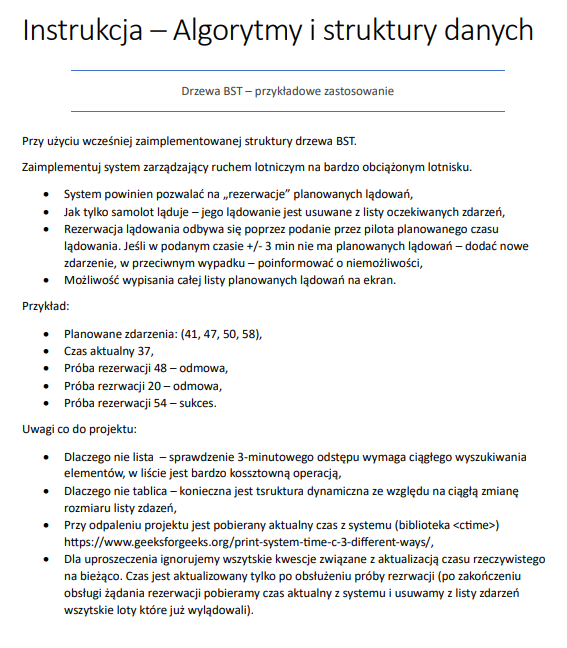
W funkcji main odbywa się proste wywołanie funkcji i przedstawienie drzewa przed i po wykonaniu funkcji.



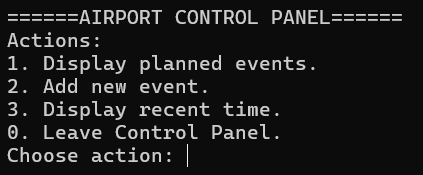
Ta funkcja służy do usuwania węzła z drzewa BST. Najpierw znajdujemy węzeł zastępujący (y), który zostanie przeniesiony na miejsce usuwanego węzła. Jeśli usuwany węzeł mający najwyżej jedno dziecko, wtedy jego następnik staje się węzłem zastępującym. Następnie ustalamy, które dziecko zastępującego węzła (x) pozostanie w drzewie. Po usunięciu węzła, jego rodzic wskazuje na nowego potomka (x), a jeśli usuwany węzeł jest korzeniem, to drzewo zostaje zaktualizowane. Na koniec, jeśli węzeł zastępujący nie jest bezpośrednio usuwanym węzłem, jego klucz jest kopiowany do miejsca usuwanego węzła.

1. **Drzewa BST - zastosowanie**

**Polecenia:**

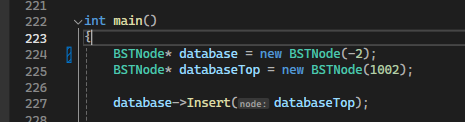


**Wynik w konsoli:**

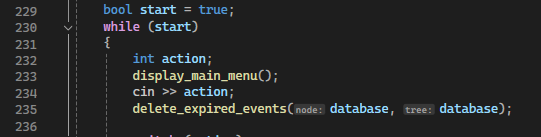


Program oparty jest o bardzo proste UI. Po wybraniu odpowiedniej cyfry wykonywane są funkcje i wyświetlane ich rezultaty.

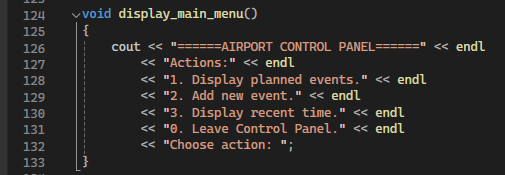
**Kod i opis rozwiązania:**



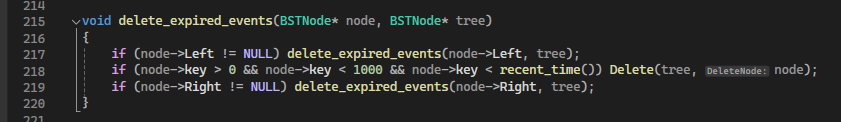
Kod w funkcji main zaczyna się od implementacji drzewa *database* i dodania elementu *databaseTop* (wartości -2 i 1002 są nieprzypadkowe. Na końcowym etapie tworzenia programu spotkałem się z problemem, który polegał na tym, że niemożliwe było dodanie elementów, które nie miałyby poprzednika lub następnika. Taki zabieg umożliwił, aby każda liczba z zakresu 1-999 miała poprzednik i następnik i aby te elementy nie sprawiały problemów związanych z za bliskim zaplanowanym wydarzeniem).



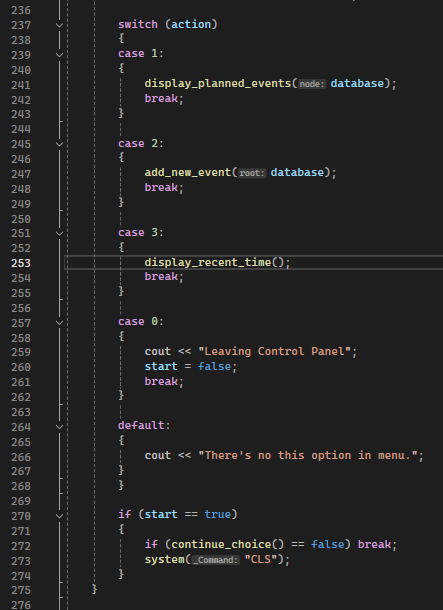
Zmienna *start* przechowuje informację o wyborze użytkownika, czy ten chce jeszcze raz uruchomić pętlę główną (while).



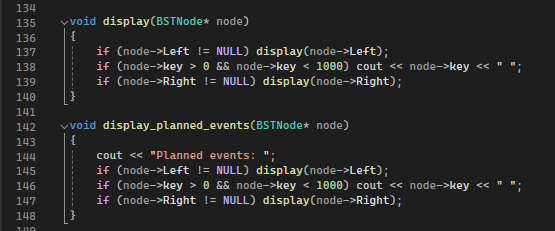
Po każdorazowym wejściu w pętlę za pomocą funkcji *display\_main\_menu* wyświetlane jest menu główne.



Potem następuje wczytanie z klawiatury użytkownika wartości do zmiennej *action*. Następnie za pomocą funkcji *delete\_expired\_events usuwane są wszystkie* przedawnione *wydarzenia.*



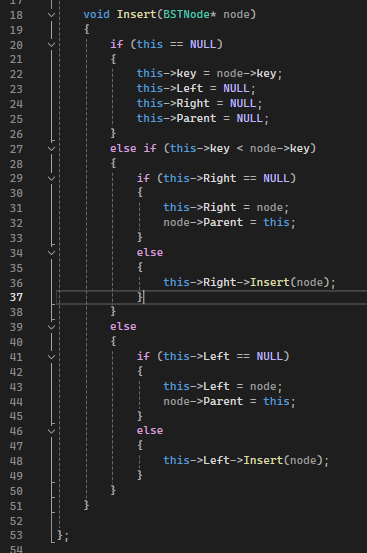
UI jest oparte o prostego switcha, za pomocą którego w zależności od wyboru użytkownika wywoływane są odpowiednie funkcje.



Funkcja *display* działa identycznie jak funkcja *PrintTreeInOrder* z pierwszej części zajęć, ale dodatkowo sprawdza zakres (1-999), aby nie były wyświetlane elementy o wartości (-2, 1002 z początku kodu). Funkcja *display\_planned\_events* powstała, aby uprościć zapis w funkcji main.



Najpierw użytkownik jest pytany o czas planowanego lądowania samolotu w zakresie od 1 do 999. Potem tworzony jest nowy węzeł drzewa BST, reprezentujący to planowane lądowanie, i dodawany do drzewa poprzez metodę *Insert*.



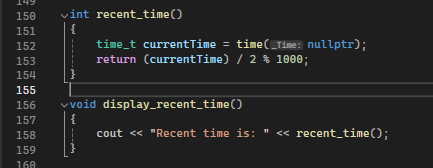
Kolejno sprawdzane jest, czy podany czas mieści się w zakresie od 1 do 999. Jeśli podany czas jest w prawidłowym zakresie, następuje sprawdzenie warunków dla akceptacji lub odrzucenia rezerwacji:

- jeśli drzewo jest puste lub zawiera tylko jeden planowany event, i czas nowego wydarzenia jest po przeszłym czasie, rezerwacja zostaje zaakceptowana.

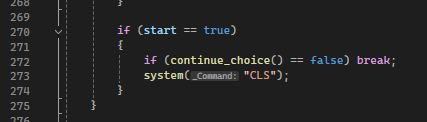
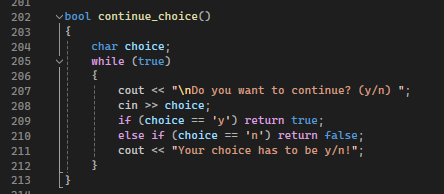
- jeśli nowy czas jest wystarczająco odległy od innych wydarzeń (co najmniej 3 jednostki czasu), rezerwacja jest akceptowana.

- W przeciwnym razie rezerwacja jest odrzucona, a nowy węzeł jest usuwany z drzewa.

Jeśli natomiast podany czas jest poza zakresem od 1 do 999, rezerwacja jest odrzucana, a użytkownik informowany jest o tym fakcie.



Funkcja *recent\_time* za pomocą funkcji z biblioteki *ctime* pobiera aktualny czas systemowy, konwertuje go w przyjazny dla użytkownika sposób i zwraca wartość liczbową. Funkcja *display\_recent\_time* powstała, aby uprościć zapis w funkcji main.

kontynuacja w funkcji main

Po każdorazowym przejściu przez główną funkcję while, o ile użytkownik nie wybrał opcji opuszczenia programu, jest on pytany, czy na pewno chce kontynuować (ten zabieg ma na celu przede wszystkim pozostawienie wyników wykonanej operacji na odpowiednią ilość czasu). Po dokonaniu wyboru usuwana jest stara, niepotrzebna już zawartość konsoli. W ten sposób całość jest zapętlona i można go w miarę możliwości płynnie używać.