// Autor: Daniel Szarek

//====================

Zawartość folderu Projekt z moim projektem zaliczeniowym z przedmiotu Algorytmy i Struktury Danych I:

- opracowanie.pdf – Ten plik z opracowaniem zadania
- main.cpp – Plik z kodem zadania napisanego w C++. W tym pliku zawarte są testy implementacji zadania oraz wypisanie testów na ekranie w terminalu.
- poly.h – Plik nagłówkowy z implementacją wielomianów na bazie tablic.
- Makefile – Do uruchomienia programu main.cpp (Makefile oferuje możliwość uruchomienia programu za pomocą komendy ‘make run’, usunięcia plików po uruchomieniu programu komendą ‘make clean’ oraz możliwość utworzenia paczki .tar.gz z zawartością foldera z zadaniem za pomocą komendy ‘make tar’).

Treść zadania:

Implementacja wielomianów na bazie tablic (szablon, współczynniki typu T). Działania na wielomianach: dodawanie (operator+), odejmowanie (operator-), mnożenie (operator*), obliczanie wartości wielomianu (algorytm Hornera), porównywanie (operator==, operator!=), wyświetlanie. Wielomiany są równe, gdy ich różnica jest wielomianem zerowym. Klasa Poly przechowuje tablicę współczynników i implementuje działania. Przydają się funkcje clear(), is_zero(), operator[] (odczyt współczynnika przy danej potędze x).

Uruchomienie:

Program można uruchomić korzystając z Makefile w folderu, w tym celu należy wpisać w terminal będąc w folderze make run.

Złożoność obliczeniowa:

Dodawanie i Odejmowanie Wielomianów: Złożoność dodawania i odejmowania wielomianów wynosi O(n), gdzie n to stopień wielomianu. Jest to spowodowane potrzebą iteracyjnego przejścia przez współczynniki obu wielomianów.

Mnożenie Wielomianów: Złożoność mnożenia wielomianów wynosi O(n^2), gdzie n to stopień większy z dwóch mnożonych wielomianów. Wynika to z faktu użycia zagnieżdżonych pętli.

Algorytm Hornera (Obliczanie Wartości Wielomianu): Złożoność algorytmu Hornera wynosi O(n), gdzie n to stopień wielomianu. Jest to efektywny sposób obliczania wartości wielomianu w danym punkcie, eliminujący potrzebę wykładniczego potęgowania.

Przypisanie i Konstruktory: Złożoność przypisania i konstruktorów jest zazwyczaj O(n), gdzie n to liczba współczynników. W przypadku kopiowania czy tworzenia nowych obiektów wymaga się iteracyjnego przeglądania tablicy współczynników.

Omówienie zadania:

Swoją implementację wielomianów utworzyłem za pomocą klasy Poly. W klasie utworzyłem dwie zmienne:

```
template <typename T>
class Poly {
    unsigned size; // Rozmiar tablicy współczynników
    T* tab; // Tablica współczynników
```

Tak jak to zostało podane w poleceniu zadanie tworzę tablicę współczynników wielomianu o typy generyczne T.

W testach main.cpp stworzyłem wielomiany na podstawie typu int.

W klasie Poly zaimplementowałem wymagane operatory oraz funkcje ponad to wzbogaciłem program o operatory działań oraz przypisania np. +=, dodałem konstruktor kopiujący i konstruktor rozmiaru oraz funkcję degree, którą wykorzystuję w testach zawartych w main.cpp. Poniżej zdjęcie wszystkich deklaracji funkcji oraz operatorów w klasie Poly zaimplementowanych dalej w pliku poly.h.

```
public:
    Poly(); // Domyślny konstruktor
    Poly(const unsigned size); // Konstruktor rozmiaru
    Poly(const T* coefficients, const unsigned size); // Konstruktor tablicy współczynników i rozmiaru
    Poly(const Poly& other); // Konstruktor kopiujący
    ~Poly(); // Destruktor
    Poly operator+(const Poly& other) const; // Operator dodawania
    Poly& operator+=(const Poly& other); // Operator dodawania i przypisania
    Poly operator-(const Poly& other) const; // Operator odejmowania
    Poly& operator-=(const Poly& other); // Operator odjęcia i przypisania
    Poly operator*(const Poly& other) const; // Operator mnożenia
    Poly& operator*=(const Poly& other); // Operator mnożenia i przypisania
    Poly& operator=(const Poly& other); // Operator przypisania
    T& operator[](const unsigned degree) const; // Operator odczytu współczynnika przy danej potędze x
    bool operator==(const Poly& other) const; // Operator równości
    bool operator!=(const Poly& other) const; // Operator nierówności
    T algorytmHornera(const T& point) const; // Obliczanie wartości wielomianu algorytmem Hornera
    void wyswietlanie() const; // Wyświetlanie wielomianu w terminalu
    void clear(); // Wyzerowanie wielomianu
    bool is_zero() const; // Sprawdzenie czy wielomian jest wielomianem zerowym
    int degree() const; // Zwrocenie stopnia wielomianu
};
```

W komentarzach zawarłem, kiedy implementuję daną deklarację np. Implementacja funkcji obliczania wartości wielomianu algorytmem Hornera.

```
// Implementacja funkcji obliczania wartości wielomianu algorytmem Hornera

template <typename T>
T Poly<T>::algorytmHornera(const T& point) const {
    T result = T();
    for (int i = size - 1; i >= 0; i--) {
        result = result * point + tab[i];
    }
    return result;
}
```

Swoją implementację zawarłem w taki sposób, że jak mamy np. tablicę współczynników {1,2,3}, to reprezentuje ona wielomian 3x^2+2x^1+1. Można to zauważyć w funkcji wyświetlanie(), gdzie zaimplementowałem wyświetlanie tego wielomianu w terminalu.

```
// Implementacja funkcji wyswietlajaca wielomian w terminalu

template <typename T>
void Poly<T>::wyswietlanie() const {
    if (is_zero()) {
        std::cout << "Wielomian pusty" << std::endl;
        return;
    }
    for (int i = size - 1; i >= 0; i--) {
        if (tab[i] != 0) {
            if (tab[i] < 0 || i == (int)size - 1) {
                std::cout << tab[i];
            } else {
                std::cout << "+" << tab[i];
            }
            if (i > 0) { std::cout << "x^" << i; }
        }
    }
    std::cout << std::endl;
}
```

Wyświetlenie wielomianów w testach:

```
void wyswietlanie_test() {
    polyDefault.wyswietlanie(); // Powinno wyświetlić: Wielomian pusty
    polySize.wyswietlanie(); // Powinno wyświetlić: Wielomian pusty
    polyCoefficients1.wyswietlanie(); // Powinno wyświetlić: 3x^2+2x^1+1
    polyCoefficients2.wyswietlanie(); // Powinno wyświetlić: 6x^2-4
}
```

Testy w pliku main.cpp określiłem jako funkcję typu bool, tak jak to miałem w zwyczaju określać na zajęciach. Wprowadziłem funkcję assert_test, która używa funkcji assert() do sprawdzania, czy test zwraca oczekiwany wynik. Wynik każdego testu będzie wypisywany na ekranie 1 dla wyniku pozytywnego, natomiast 0 dla wyniku negatywnego testu. W przypadku niepowodzenia testu program zakończy się niepowodzeniem z komunikatem.

Zdjęcie kompilacji programu main.cpp:

```
PS C:\Users\dszar\OneDrive\Pulpit\GitHub\AlgorytmyiStrukturyDanychI2023\Projekt> make run
g++ -Wall -std=c++11 -c main.cpp -o main.o
g++ -o main.x -Wall -std=c++11 main.o
./main.x
Witaj w programie implementacji wielomianu na bazie tablic!
(Ponizej zostana przeprowadzone testy wynik 1 to true, 0 to false)

Test: konstruktora domyslnego : 1
Test: konstruktora rozmiaru : 1
Test: konstruktora tablicy wspolczynnikow i rozmiaru : 1
Test: konstruktora kopiujacego : 1
Test: operatora + : 1
Test: operatora += : 1
Test: operatora - : 1
Test: operatora -= : 1
Test: operatora * : 1
Test: operatora *= : 1
Test: operatora = : 1
Test: operatora [] : 1
Test: operatora = (kopiujacy) : 1
Test: operatora != : 1
Test: algorytmuHornera : 1
Test: clear : 1
Test: is_zero : 1
Wielomian pusty
Wielomian pusty
3x^2+2x^1+1
6x^2-4
PS C:\Users\dszar\OneDrive\Pulpit\GitHub\AlgorytmyiStrukturyDanychI2023\Projekt>
```