Projekt Algorytmy

Opis programu:

Program zawiera kod służący do wykonywania na wielomianach operacji:

- Dodawania (+)
- Odejmowania (-)
- Mnożenia (*)
- Obliczania wartości wielomianu schematem Hornera

Jak wprowadzać dane:

Użytkownik wpisuje w programie współczynniki wielomianów w postaci tablicy, oraz ich wielkość w postaci liczby całkowitej (najwyższa potęgą jaką przyjmie wielomian):

```
//tablice z wartosciami jakie chcemy nadac wielomianom (wspolczynniki od najnizszego do najwyzszego)
int coef1[] = { [0]: 2,  [1]: 8,  [2]: 3};
int coef2[] = { [0]: 7,  [1]: 1,  [2]: 2,  [3]: 6};

//Tworzenie wielomianow, podajemy tablice z ich wspolczynnikami oraz "power" czyli potege do ktorej beda dochodzic
Polynomial<int> p1( coeff: coef1,  power: 2);
Polynomial<int> p2( coeff: coef2,  power: 3);
```

Użytkownik musi również podać wartość dla jakiej ma być obliczona wartość wielomianu przy użyciu schematu Hornera:

```
cout << "Wartosc wielomianu schematem hornera: " << endl;
cout << "Wartosc wielomianu W1(x) dla x = 7: " << p1.horner( x: 7) << "\n";
cout << "Wartosc wielomianu W2(x) dla x = 5: " << p2.horner( x: 5) << "\n";
```

Składowe programu:

```
template <typename T>
class Polynomial {
    T *coeff; //wspolczynniki
    int power; //najwyzsza potega
public:

    Polynomial(int _power) : power(_power) {
        coeff = new T[power + 1];
        for (int i = 0; i < power + 1; ++i) {
            coeff[i] = T();
        }
    }

    Polynomial(T *_coeff, int _power) : power(_power) {
        coeff = new T[power + 1];
        for (int i = 0; i < power + 1; ++i) {
            coeff[i] = _coeff[i];
        }
    }

    //desktruktor
    ~Polynomial() {
        power = 0;
        delete[] coeff;
    }
```

Klasa Polynomial jest szablonem w którym możemy przechowywać wielomiany o różnych typach współczynników.

W klasie mamy zdefiniowane następujące konstruktory:

1. Pierwszy konstruktor przyjmuje jeden argument typu int _power, która oznaczać będzie najwyższą potęgę wielomianu. Tworzy on dynamiczną tablicę o długości o 1 większej niż podana potęga a dla każdego elementu przypisuje wartość domyślną T.

2. Drugi konstruktor przyjmuje dwa argumenty: wskaźnik do tablicy _coeff która zawiera współczynniki i liczbę typu int _power, która tak jak w konstruktorze wyżej oznacza najwyższą potęgę wielomianu. Tworzy on nową dynamiczną tablicę o długości o 1 większej od _power i przypisuje każdemu elementowi wartość z tablicy _coeff.
3. Klasa zawiera też destruktor usuwający tablice oraz zerujący zmienną power.

```
//operator dodawania
Polynomial<T> operator+(const Polynomial<T> &other) {
    int n1 = this->power;
    int n2 = other.power;
    int higher = max(n1, n2);
    Polynomial<T> result(higher);

    for (int i = 0; i <= higher; ++i) {
        T coef1;
        T coef2;
        if (i <= n1) {
            coef1 = this->coeff[i];
        } else {
            coef1 = T();
        }
        if (i <= n2) {
            coef2 = other.coeff[i];
        } else {
            coef2 = T();
        }
        result.coeff[i] = coef1 + coef2;
    }
    return result;
}
```

Powyższy kod przedstawia operację dodawania dla wielomianów. Zaczyna się od określenia zmiennych n1 i n2 które będą przechowywać najwyższą potęgę dla każdego z wielomianów. Następnie za pomocą zmiennej higher (wartość najwyższej potęgi z dwóch wielomianów) tworzony jest wielomian result. W pętli for przechodzimy przez wszystkie indeksy (od 0 do higher), określamy dwie zmienne coef1 i coef2 które będą przechowywać współczynnik dla danego indeksu dla każdego z dwóch wielomianów. Jeśli indeks jest mniejszy niż n1 lub n2 (w zależności czy mówimy o zmiennej coef1 czy coef2), zmienna jest ustawiona na odpowiedni współczynnik dla tego indeksu. W przeciwnym wypadku zmienna zostanie ustawiona na T(). Na sam koniec wielomian result jest zwracany jako wynik operacji.

```
Polynomial operator-(const Polynomial &other) {
    int n1 = this->power;
    int n2 = other.power;
    int higher = max(n1, n2);
    T coef[higher + 1];
    Polynomial<T> result(coef, higher);

    for (int i = 0; i <= higher; i++) {
        T c1;
        T c2;
        if (i <= n1) {
            c1 = this->coeff[i];
        } else {
            c1 = 0;
        }
        if (i <= n2) {
            c2 = other.coeff[i];
        } else {
            c2 = 0;
        }
        result.coeff[i] = c1 - c2;
    }
    return result;
}
```

Kolejna jest operacja odejmowania. Podobnie jak w operacji dodawania korzystamy z deklaracji zmiennych: n1, n2 i zmiennej higher służącej do stworzenia wielomianu wynikowego. Następnie w pętli kolejno ustalany jest wyższy stopień między dwoma wielomianami, przez który następnie jest iterowane i odejmowane są odpowiednie współczynniki obydwu wielomianów. W przypadku gdy indeks i jest większy od stopnia wielomiany współczynnik jest ustawiany na 0. Wynik jest zwracany jako wielomian result.

```
Polynomial operator*(const Polynomial &other) {
    int n1 = this -> power;
    int n2 = other.power;
    int result_n = n1 + n2;

    Polynomial<T> result(result_n);

    for (int i = 0; i < n1 + 1; ++i) {
        for (int j = 0; j < n2 + 1; ++j) {
            result.coeff[i + j] += this -> coeff[i] * other.coeff[j];
        }
    }
    return result;
}
```

Podobnie jak w operacjach powyżej korzystamy ze zmiennych n1 i n2, ale w tym wypadku najwyższy stopień wielomianu result przyjmie sumę stopni wielomianów. Potem korzystając z pętli zagnieżdżonych iterujemy po wszystkich współczynnikach dwóch wielomianów i dodawaniu ich do siebie na odpowiedniej potędzę wielomianu wyjściowego. Zwracany jest wielomian result

```
T horner(T x){
    T result = 0;
    for (int i = power; i >= 0; --i) {
        result = result * x + coeff[i];
    }
    return result;
}
```

Kolejnym fragmentem tekstu jest funkcja służąca do obliczania wartości wielomianu za pomocą schematu hornera. Dla każdego indeksu od “power” do zera, mnożymy result przez x i dodajemy kolejny element tablicy coeff[]. Na samym końcu zwracana zostaje wartość obliczonej sumy.

```
void display() {
    bool t = true;
    for (int i = power; i >= 0; --i) {
        if (coeff[i] != 0) {
            if (!t) {
                if (coeff[i] > 0) {
                    cout << " + ";
                } else {
                    cout << " - ";
                }
            }
            t = false;
            cout << abs(coeff[i]);
            if (i > 1) {
                cout << "x^" << i;
            } else if (i == 1) {
                cout << "x";
            }
        }
    }
    if (t) {
        cout << 0;
    }
    cout << endl;
}
```

Ostatni element kodu służy do wyświetlania na ekranie wielomianu. Przyjmuje on jako argumenty tablice współczynników i stopień wielomianu. Używamy zmiennej “t” typu bool która pomaga nam w drukowaniu wielomianu na ekran. Pętla for służy do sprawdzania znaku współczynnika i wypisywania ich kolejno ze znakami “+” lub “-”. Następnie sprawdzany jest stopień wielomianu i w zależności od tego drukowane jest “x^” lub samo “x”.Pętla działa aż do zejścia do najmniejszej potęgi wielomianu. Na sam koniec sprawdzane jest czy dalej zmienna t nie jest ustawiona na “true”, jeśli tak by było wypisane zostanie 0.

Złożoność czasowa:

- Operator dodawania (+) i odejmowania (-) wykonuje się w czasie liniowym O(n), gdzie n to wykładnik najwyższej potęgi wielomianów biorących udział w operacji.

- Operator mnożenia (*) wykonuje się w czasie O(n^2), gdzie n to suma wykładników najwyższych potęg wielomianów biorących udział w operacji.

- Funkcja horner() wykonuje się w czasie liniowym O(n), gdzie n to wykładnik najwyższej potęgi wielomianu.

- Funkcja display() wykonuje się w czasie liniowym O(n), gdzie n to wykładnik najwyższej potęgi wielomianu.