Wojciech Ładyga - projekt zaliczeniowy ONP

Język / technologia: C++

Kompilacja / uruchomienie

```
//kompilacja do main
g++ -o main main.cpp

//uruchomienie
main - windows
./main - linux
```

Program ten po uruchomieniu umożliwia 3 podstawowe działania

1. konwersje z postaci infiksowej na ONP
2. wyliczenie wyrażenia podanego w ONP
3. wyliczenie wyrażenia podanego w notacji infiksowej przez konwersje do onp i następnie wyliczenie wartości

Operacje, które są obsługiwane to `-, +, *, /, ^`
Również umożliwia podanie wyrażeń z nawiasami okrągłymi `()`

Program zawiera 2 podstawowe klasy

- 1. klasa obsługująca stos
- 2. klasa obsługująca konwersje i wyliczanie wartości wyrażeń

Najważniejszą klasą w tym projekcie jest klasa Stos. Implementacja tej klasy pozwala nam korzystać z onp.

ad 1. `Klasa Stos`
Jest to stos w postaci tablicowej i zawiera następujące operacje

```
position Top; //szczyt stosu
void Push(elementtype x); //dodaj el na szczyt
void Pop(); //usuń el ze szczytu
bool Empty(); //sprawdza czy pusty
elementtype TopElem(); //zwróć el z wierzchu
void Makenul(); //uczyń stos pustym
```

ad 2. `Klasa Onp`

```
bool sprOperator(string &znak); //sprawdza czy podany znak jest operatorem
bool sprOpZnawiasami(string &znak); //sprawdza czy jest operatorem i nawiasem
int priorytet(const string &znak); //metoda zwraca odpowiednie priorytety operacji
```

```
void normalToOnp(); //metoda zamienia z infiksowej do ONP
elementtype onpCount();  //metoda zwraca wartość wyliczoną w onp
```

Opis Kodu `klasy Onp`:

```
//sprawdzamy czy dany znak jest -,+,/,*,^
bool Onp::sprOperator(string &znak)
{
    return (znak == "+" || znak == "-" || znak == "*" || znak == "/" || znak ==
"^");
}

//sprawdzamy znaki jak metodą wcześniejszą oraz czynie mamy do czynienia z
nawiasem
bool Onp::sprOpZnawiasami(string &znak)
{
    bool znakSpr = false;
    for(int i = 0; i < znak.size(); i++)
    {
        //kowertujemy chara do stringa
        char tmpZnak = znak[i];
        string tmp;
        tmp = tmpZnak;

        if(sprOperator(tmp) || tmp == "(" || tmp == ")")
        {
            znakSpr = false;
        }
        else
        {
            znakSpr = true;
        }
    }
    return znakSpr;
}

//metoda odpowiedzialna za ustawienie priorytetu operacji
int Onp::priorytet(const string &znak)
{
    if(znak == "^")
    {
        return 3;
    }
    if(znak == "*" || znak == "/")
    {
        return 2;
    }
    if(znak == "+" || znak == "-")
    {
```

```
        return 1;
    }
    else
    {
        return 0;
    }
}

//Konwersja wyrażenia z notacji tradycyjnej do ONP
void Onp::normalToOnp(){
    Stos s;
    int wielkosc = rownanieTab.size();
    int i = 0;

    //przechodzimy przez całe wyrażenie
    while(wielkosc != 0){
        //pobieramy el
        string el = rownanieTab[i];

        //jeśli nie mamy znaków operacji i nawiasów to wrzucamy naszą wartość do
wektora, który przechowuje nasze wyrażenie w posatci onp
        if(sprOpZnawiasami(el)){
            wyjscie.push_back(el);
        }
        if(el == "(")
        {
            s.Push(el);
        }

        //jeśłi mamy nawias domukający to
        if(el == ")")
        {
            while(!s.Empty() && s.TopElem() != "(")
            {
                wyjscie.push_back(s.TopElem());
                s.Pop();
            }
            s.Pop();
        }

        //jeśli mamy do czynienia z operatorem
        if(sprOperator(el) == true)
        {
            //sprawdzamy odpowiednio wagi i wrzucamy do wektora usuwając el ze
stosu
            while(!s.Empty() && priorytet(s.TopElem()) >= priorytet(el))
            {
                wyjscie.push_back(s.TopElem());
                s.Pop();
            }
            s.Push(el);
```

```
        }

        i++;
        wielkosc--;
    }

    //na końcu czyścimy stos i wrzucamy ostatni el do wektora
    while(!s.Empty())
    {
        wyjscie.push_back(s.TopElem());
        s.Pop();
    }
    copy(wyjscie.begin(), wyjscie.end(), back_inserter(tmpWyjscie));

}

//zracamy wynik podany w onp jako el znajdujący sie na szczycie stosu
elementtype Onp::onpCount(){
    Stos s;
    double wynik;
    int length = onp.size();
    for (int i = 0; i < length; i++) {

    if(sprOperator(onp[i])){

            //pobieramy 2 el ze stosu jeśli natrafimy na operator
            string znak1 = s.TopElem();
            double char1 = atoi(znak1.c_str()); //konwersja stringa do doubla
            s.Pop();
            string znak2 = s.TopElem();
            double char2 = atoi(znak2.c_str());
            s.Pop();

            //sprawdzamy odpowiednie wyrażenie i wykonujemy operacje
            if(onp[i] == "*"){
                wynik = char2*char1;
            }
            if(onp[i] == "+"){
                wynik = char2+char1;
            }
            if(onp[i] == "/"){
                wynik = char2/char1;
            }
            if(onp[i] == "-"){
                wynik = char2-char1;
            }
            if(onp[i] == "^"){
                wynik = pow(char2, char1);
            }

            //następnie doubla wrzucamy do stringa i wrzucamy na stos
```

```
            ostringstream ss;
            ss << wynik;
            string tmp = ss.str();
            s.Push(tmp);
        }else{
            s.Push(onp[i]);
        }
    }

    //zwracamy wynik
    return s.TopElem();
}
```

Operacje oparte na klasie Onp odpalane są w pliku main.cpp

plik ten zawiera metody

```
    void tradycyjnaToOnp(Onp on); //konwertuje równanie podane w postaci
infiksowej do onp
    void wyliczOnp(Onp on); //wylicza wartość podaną w onp
    void liczymy(Onp on); //liczy podaną wartość z notacji infiksowej konwertując
do onp i wyliczając wartość
    void czysc(); //czysci ekran
    void menu(Onp on); //wyświetla menu
```

Część kodu znajdującego się w pliku `main.cpp`

```
//funkcja konwertująca do onp korzystając z metody normalToOnp() z klasy Onp w
celu przekonwertowania wyrażenia
void tradycyjnaToOnp(Onp on){
    cout << "Podaj rownanie w notacji tradycyjnej (kazdy el oddzielony spacja) "
<< endl;
    cout << ">";

    //pobiera el podane w lini jako jeden string
    getline(cin, rownanie);

    //string zostaje wrzucony do wektora
    //rozdzielenie stringa pozwala zachowanie spacji między wyrażeniami
    istringstream iss(rownanie);
    string s;
    while ( getline( iss, s, ' ' ) ) {
        rownanieTab.push_back(s);
    }

    on.normalToOnp();
    for(int i = 0; i < wyjscie.size(); i++)
```

```
    {
        cout<<wyjscie[i];
    }
}

//funkcja wylicza wyrażenie podane w opn korzystając z metody onpCount z klasy Onp
void wyliczOnp(Onp on){
    cout << "Podaj rownanie w onp (kazdy el oddzielony spacja)" << endl;
    cout << ">";
    getline(cin, rownanie);

    //string to vector
    istringstream iss(rownanie);
    string s;
    while ( getline( iss, s, ' ' ) ) {
        onp.push_back(s);
    }

    //onpTonormal();
    cout << on.onpCount();
}

//funkcja konwertuje podane wyrażenie w notacji infiksowej i wylicza wyrażenie za
pomocą onp
void liczymy(Onp on){
    cout << "Podaj rownanie w notacji tradycyjnej (kazdy el oddzielony spacja) "
<< endl;
    cout << ">";
    getline(cin, rownanie);

    //string to vector
    istringstream iss(rownanie);
    string s;
    while ( getline( iss, s, ' ' ) ) {
        rownanieTab.push_back(s);
    }
    on.normalToOnp();
    copy(on.tmpWyjscie.begin(), on.tmpWyjscie.end(), back_inserter(onp));
    cout << on.onpCount();
}
```