ONP - Odwrotna Notacja Polska

ONP (ang. RPN - Reverse Polish Notation) jest sposobem zapisu wyrażeń arytmetycznych bez stosowania nawiasów, w którym symbol operacji występuje po argumentach. Poniżej podajemy przykłady wyrażeń w ONP:

Notacja normalna	ONP
2 + 2 =	2 2 + =
3 + 2 * 5 =	3 2 5 * + =
2 * (5 + 2) =	2 5 2 + * =
(7 + 3) * (5 - 2) ^ 2 =	7 3 + 5 2 - 2 ^ * =
4 / (3 - 1) ^ (2 * 3) =	4 3 1 - 2 3 * ^ / =

ONP jest powszechnie stosowana w kompilatorach języków HLL do obliczania wartości wyrażeń arytmetycznych. Spowodowane jest to prostotą obliczeń w porównaniu do zwykłej notacji z nawiasami. Algorytm obliczania wartości wyrażenia ONP wykorzystuje stos do składowania wyników pośrednich. Zasada pracy tego algorytmu jest bardzo prosta:

Z wejścia odczytujemy kolejne elementy wyrażenia. Jeśli element jest liczbą, zapisujemy ją na stosie. Jeśli element jest operatorem, ze stosu zdejmujemy dwie liczby, wykonujemy nad nimi operację określoną przez odczytany element, wynik operacji umieszczamy z powrotem na stosie. Jeśli element jest końcowym znakiem '=', to na wyjście przesyłamy liczbę ze szczytu stosu i kończymy. Inaczej kontynuujemy odczyt i przetwarzanie kolejnych elementów.

Dla przykładu obliczmy tą metodą wartość wyrażenia **7 3 + 5 2 - 2 ^ * =**

we	stos	wy
7	7	
3	7 3	
+	10	
5	10 5	
2	10 5 2	

-	10 3	
2	10 3 2	
^	10 9	
*	90	
=		90

Algorytm obliczania wartości wyrażenia w ONP

Dane wejściowe

we - wyrażenie ONP, które można odczytywać poszczególnymi elementami. Na końcu wyrażenia znajduje się znak '='

Dane wyjściowe

wartość wyrażenia ONP

Dane pomocnicze

stos - przechowuje wyniki pośrednie
el - element odczytany z we
a,b - przechowuje dane ze stosu
w - wynik operacji arytmetycznej

Lista kroków

Krok 1: **Czytaj** *el* z *we*
Krok 2: **Jeśli** *el* nie jest liczbą, **to idź** do kroku 5
Krok 3: **Umieść** *el* na stosie
Krok 4: **Idź** do kroku 1
Krok 5: **Jeśli** *el* jest znakiem '=', **to idź** do kroku 10
Krok 6: **Pobierz** ze stosu dwie liczby *a* i *b*
Krok 7: **Wykonaj** nad liczbami *a* i *b* operację określoną przez *el* i umieść wynik w *w*
Krok 8: **Umieść** *w* na stosie
Krok 9: **Idź** do kroku 1
Krok 10: **Prześlij** na wyjście zawartość wierzchołka stosu
Krok 11: **Zakończ**

Przykładowe dane dla programu

Uwaga: dane liczbowe nie mogą być ujemne. Dozwolone operatory to +, -, *, / i ^. Program nie ma zaimplementowanej obsługi błędów. Wprowadzane wyrażenie ONP musi być poprawne składniowo, inaczej program nie będzie zwracał poprawnych wyników.

77 3 + 5 2 - * =

```
#include <iostream>

using namespace std;

int main()
{
    int  stos[256], ws = 0;
    int  a, b, w;
    char el[256];

    do
    {
        cin >> el;

        if(el[0] >= '0' && el[0] <= '9')
        {
            a = b = 0;
            do a = 10 * a + el[b++] - 48; while(el[b]);
            stos[ws++] = a;
        }
        else if(el[0] == '=') cout << stos[--ws] << endl;
        else
        {
            b = stos[--ws]; a = stos[--ws];

            switch(el[0])
            {
                case '+': w = a + b; break;
                case '-': w = a - b; break;
                case '*': w = a * b; break;
                case '/': w = a / b; break;
                case '^': w = 1;
                          while(b--) w *= a;
                          break;
            }

            stos[ws++] = w;
        }
    } while(el[0] != '=');

    return 0;
}
```

Przekształcanie wyrażeń na ONP

Algorytm obliczania wartości wyrażenia ONP wykorzystywał stos do składowania wyników pośrednich. Algorytm przekształcania wyrażeń arytmetycznych na ONP będzie wykorzystywał stos do przechowywania operatorów i nawiasów. Ponieważ są to znaki, stos operatorów będzie zrealizowany w postaci tablicy znakowej.

Operatory arytmetyczne posiadają różne priorytety (różną wartość). Jeśli w wyrażeniu arytmetycznym spotka się kilka operatorów, to operatory z wyższym priorytetem będą wykonane najpierw:

2 + 5 * 3 - 18 / 3 ^ 2 = 2 + 5 * 3 - 18 / 9 = 2 + 15 - 2 = 15

Operatory o równym priorytecie są wykonywane z lewa na prawo

Określmy priorytety operatorów następująco:

Op.	**Pr.**
^	3
* /	2
+ -	1
(	0

Operacje potęgowania mają najwyższy priorytet - będą wykonywane przed innymi operacjami. Najniższy priorytet mają operacje dodawania i odejmowania - będą wykonywane po wszystkich innych działaniach.

Zasada przekształcania wyrażenia na ONP jest następująca:

Czytamy z wejścia element. Dalsze postępowanie zależy od rodzaju elementu:

1. Liczba - przesyłamy ją na wyjście.
2. Operator - przesyłamy ze stosu na wyjście wszystkie operatory o priorytecie wyższym lub równym odczytanemu operatorowi. Operator dopisujemy do stosu.
3. (- nawias otwierający zawsze dopisujemy do stosu.
4.) - przesyłamy ze stosu na wyjście wszystkie operatory aż do napotkania nawiasu otwierającego. Nawias ten usuwamy ze stosu.
5. = - przesyłamy ze stosu na wyjście wszystkie pozostałe na nim operatory wraz ze znakiem '=' i kończymy algorytm

Wracamy na początek do odczytu elementu wyrażenia.

Dla przykładu obliczmy tą metodą wartość wyrażenia **(15 - 3) ^ (3 + 2) * 6 / 3 =**

we	stos	wy
(	(	
15	(	15
-	(-	15
3	(-	15 3
)		15 3 -
^	^	15 3 -
(	^ (	15 3 -
3	^ (	15 3 - 3
+	^ (+	15 3 - 3
2	^ (+	15 3 - 3 2
)	^	15 3 - 3 2 +
*	*	15 3 - 3 2 + ^
6	*	15 3 - 3 2 + ^ 6
/	/	15 3 - 3 2 + ^ 6 *
3	/	15 3 - 3 2 + ^ 6 * 3
=		15 3 - 3 2 + ^ 6 * 3 / =

Algorytm przeliczania na ONP

Dane wejściowe

we - wyrażenie arytmetyczne zakończone znakiem =

Dane wyjściowe

wy - postać ONP wyrażenia wejściowego

Dane pomocnicze

stos - przechowuje operatory
el - element odczytany z we

Lista kroków

Krok 1: **Czytaj** z *we* do *el*
Krok 2: **Jeśli** *el* jest liczbą, **to idź** do kroku 17
Krok 3: **Jeśli** *el* = '=', **to idź** do kroku 15
Krok 4: **Jeśli** *el* = '(', **to idź** do kroku 12

Krok 5: **Jeśli** *el* = ')', **to idź** do kroku 9

Krok 6: **Dopóki** na stosie jest operator o wyższym lub równym priorytecie od *el*, przesyłaj operator ze stosu na *wy*.

Krok 7: **Dopisz** do stosu *el*

Krok 8: **Idź** do kroku 1

Krok 9: **Dopóki** na stosie jest element różny od '(', przesyłaj operator ze stosu na wy.

Krok 10: **Usuń** '(' ze stosu

Krok 11: **Idź** do kroku 1

Krok 12: **Dopisz** '(' do stosu

Krok 13: **Idź** do kroku 1

Krok 14: **Dopóki** stos coś zawiera, przesyłaj operator ze stosu na wyjście

Krok 15: **Prześlij** na wyjście '='

Krok 16: **Zakończ**

Krok 17: **Prześlij** *el* na wyjście

Krok 18: **Idź** do kroku 1

Przykładowe dane dla programu

Uwaga: dane liczbowe nie mogą być ujemne. Dozwolone operatory to +, -, *, /, ^ oraz nawiasy okrągłe. Każdy element wyrażenia musi być oddzielony spacją od elementów sąsiednich. Program nie posiada obsługi błędów i wyrażenie wejściowe musi być poprawne.

((5 - 2) ^ (3 + 1) / (2 + 1) + 12) * 21 =

```cpp
#include <iostream>

using namespace std;

int main()
{
    char stos[256],el[256];
    int ws = 0;

    do
    {
        cin >> el;

        if(el[0] >= '0' && el[0] <= '9') cout << el << " ";
        else
            switch(el[0])
            {
                case '+': ;
                case '-': while(ws && stos[ws - 1] != '(') cout << stos[--ws] << " ";
                          stos[ws++] = el[0];
                          break;
                case '*': ;
                case '/': while(ws && stos[ws - 1] != '(' &&
                                      stos[ws - 1] != '+' &&
                                      stos[ws - 1] != '-') cout << stos[--ws] << " ";
                          stos[ws++] = el[0];
                          break;
                case '^': while(ws && stos[ws - 1] == '^') cout << stos[--ws] << " ";
                          stos[ws++] = el[0];
                          break;
                case '(': stos[ws++] = '(';
                          break;
                case ')': while(stos[ws - 1] != '(') cout << stos[--ws] << " ";
                          ws--;
                          break;
                case '=': while(ws) cout << stos[--ws] << " ";
                          cout << " =\n\n";
                          break;
            }

    } while(el[0] != '=');

    return 0;
}
```

Dla ambitnych: na podstawie tych dwóch programów stwórz program, który oblicza wartość wyrażeń arytmetycznych zawierających operatory +, -, *, /, ^ oraz nawiasy okrągłe.