Odwrotna notacja polska

Generalnie, mamy trzy sposoby zapisu wyrażeń matematycznych. Standardowym dla nas jest umieszczanie znaku operacji pomiędzy operandami – to nazywamy notacją ***in****fiksową.* Istnieje również notacja ***pre****fiksowa*, częściej używana w logice, w której znak umieszczany jest przed operatorami. Zwana jest ona również notacją polską lub notacją Łukasiewicza.

Dzisiaj skupimy się jednak na odwrotnej notacji polskiej, inaczej ONP lub notacja ***post****fiksowa.* W tym zapisie znak zapisywany jest po operandach. Pozwala to na całkowitą rezygnację z korzystania z nawiasów przy równoczesnym zachowaniu kolejności wykonywania obliczeń. Autor, Australijczyk, zaproponował, aby nazwać ten sposób zapisu "Azciweisakul notation". Bo „Lukasiewicza" od tyłu. Dobrze, że się nie przyjęło.

Jeśli chodzi o wszelkie przeliczanie, to skorzystamy z dwóch bardzo przyjemnych algorytmów. Zaczniemy od tego, jak przeliczyć z notacji infiksowej na postfiksową.

Wczytujemy dane po kolei. Jeżeli wczytany element jest:

- **Zmienną (stałą)**
  - Przesyłamy go na wyjście
- **(**
  - Przesyłamy go na stos
-)
  - Odczytujemy ze stosu i przepisujemy na wyjście wszystkie operatory aż do (, którego nie przepisujemy
- **+, -, *, /, ^**
  - Jeżeli priorytet operatora wczytywanego jest **wyższy** od priorytetu tego na wierzchu stosu (lub stos jest pusty), dopisujemy na stos.
  - Jeżeli priorytet operatora wczytywanego jest **mniejszy lub równy** od tego na wierzchu stosu, odczytujemy i przepisujemy na wyjście kolejne operatory z wierzchu stosu, dopóki ich priorytet jest >= wczytanego, po czym wpisujemy na stos operator.

Po wczytaniu wszystkich elementów przepisujemy ze stosu na wyjście pozostałe operatory

Operator	Priorytet
(	0
+ -)	1
x /	2
^	3

Po lewej znajduje się tabelka z priorytetami operatorów. W powyższym algorytmie pojawia się jeszcze określenie „stos", więc pewnie przydałoby się wyjaśnić, o co z nim chodzi. Stos to taka struktura danych, w której dane są umieszczane i pobierane ze szczytu. Inaczej bufor **LIFO**, czyli last in – first out. Można to porównać do kupki książek. Ostatnia, którą na tej kupce położymy będzie pierwszą, którą z niego zdejmiemy. Przechodząc do zastosowania stosu w wykonaniu

algorytmu, bo brzmi to na skomplikowane, ale tak naprawdę to nie jest. Dla przykładu, obliczymy $(12 + 8) - 2 * 6$

Przypomnę, że notacja postfiksowa oznacza, że znak występuje po operandach. I w ostatecznym wyniku to widać doskonale.

12 8 + 2 6 * -

Zachowując kolejność wykonywania obliczeń, której nauczyliśmy się w okolicach 2 klasy podstawówki wiemy, że bierzemy sobie 12 i 8, dodajemy do siebie, wychodzi 20. Następnie bierzemy 2 i 6, mnożymy przez siebie, wychodzi 12. Na końcu bierzemy nasze wyniki, czyli 20 i 12 i odejmujemy, wychodzi 8.

Step	Input	Stack	Output
1	(	(	
2	12	(	12
3	+	(+	12
4	8	(+	12 8
5	)		12 8 +
6	-	-	12 8 +
7	2	-	12 8 + 2
8	*	- *	12 8 + 2
9	6	- *	12 8 + 2 6
10			12 8 + 2 6 * -

W drugą stronę jest jeszcze prościej, ale nie będziemy przechodzić z ONP na infix, wyliczymy od razu wynik. Algorytm przedstawia się następująco:

Dla wszystkich elementów wykonujemy:

- Jeżeli wczytany symbol jest liczbą, wkładamy go na stos
- Jeżeli jest operatorem, to:
    - Ze stosu zdejmujemy jeden element (a)
    - Ze stosu zdejmujemy drugi element (b)
    - Na stosie umieszczamy wynik b *operator* a

Po wykonaniu dla wszystkich symboli w ONP na stosie pozostanie nam wynik wyrażenia. Dla przykładu, na poprzednim przykładzie, czyli **12 8 + 2 6 * -**

Step	Input	Stack
1	12	12
2	8	12 8
3	+	20
4	2	20 2
5	6	20 2 6
6	*	20 12
7	-	8

W tej wizualizacji stos rośnie „w prawo”, czyli kolejne elementy na stos wrzucam po prawej stronie. W kroku 5 na wierzchu stosu znajduje się 6, a na spodzie 20. Możecie się spotkać w internecie również z odwrotnym zapisem, kolejne elementy będą dokładane z lewej strony. Na szczęście w większości z nich spód stosu alignowany jest do linii rozdzielającej komórki

Zadanka z ONP:

- Zapisz w ONP: $2a + 4b$.
- Zapisz w ONP: $7 + 3 + ab + 1$
- Oblicz w ONP: $3, 4, 22, 2, +, -,\ /$

~~Sztos~~ Stos w Asm

Stos określany jest w asemblerze rejestrem SP – stack pointer. Tak naprawdę, to określany jest parą SS:SP, czyli *stać segment: stack pionter*. I tu się pojawia pierwsza trudność, bowiem w architekturze x86 stos nie rośnie, a maleje. A raczej rośnie w dół. To znaczy, że gdy przy domyślnym uruchomieniu na rejestrach znajduje nam się:

```
SS = 0D25
SP = FFFE
```

To w momencie, w którym umieścimy na wierzchu stosu wartość FFFF, to adresy 0D25:FFFE i 0D25:FFFF pozostaną nienaruszone, nasza wartość zostanie umieszczona na 0D25:FFFC-D. Po umieszczeniu na stosie wartości AAAA, ta zostanie umieszczona na 0D25:FFFA-B:

```
            0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0D25:FFE0  00 00 00 00 00 00 00 00 0E 01 AF A1 15 A1 25 0D
0D25:FFF0  02 72 AA AA 09 01 25 0D 87 52 AA AA FF FF 00 00
```

Powyższy opis przedstawia wykonanie instrukcji

```
push 0xFFFF
push 0xAAAA
```

Rejestr SP odpowiada zawsze za wskazywanie szczytu stosu. Albo spodu, zależy od której strony patrzymy. W każdym razie – SP wskazuje na to miejsce, gdzie dokładamy wartości. Instrukcja *push* wymaga podania dowolnej wartości w rozmiarze *word.* Może to zatem być zarówno rejestr, sztywna wartość, adres w pamięci, generalnie do wyboru do koloru.

Aby zdjąć ze stosu wartości – a przypomnę, że zawsze ściągamy ze szczytu – skorzystamy z instrukcji *pop.* Ta wymaga już wskazania miejsca docelowego dla ściąganej ze stosu wartości, może to być rejestr ogólnego przeznaczenia lub segmentowy, alternatywnie adres w pamięci.

Są jeszcze instukcje, które pozwalają wrzucić oraz zdjąć ze stosu w takiej samej kolejności rejestry. Te instrukcje to:

```
pusha
popa
```

Pusha umieszcza na stosie zawartość rejestrów w kolejności AX, CX, DX, BX, SP, BP, SI, DI. *Popa* natomiast zdejmuje wartości ze stosu i umieszcza je po kolei w rejestrach DI, SI, BP, SP, BX, DX, CX, AX. Można w takim razie powiedzieć, że ta kombinacja zachowa nam zawartość rejestrów abyśmy mogli w nich namieszać, a potem bez konsekwencji przywróci je nam do pierwotnego stanu – pod warunkiem, że pomiędzy nimi nie dodamy niczego na stos bez uprzedniego zdejmowania. Używamy tej kombinacji zazwyczaj przy wykonywaniu funkcji, aby po wyjściu rejestry były dokładnie takie same jak przed wejściem do niej.

Zadania na dziś

Na plusika poproszę o zestaw 5 spośród zamieszczonych poniżej 7 zadanek. Każde z nich wymagać będzie wyliczenia sobie ręcznie zapisu w ONP oraz napisania odpowiedniego programu. Proponuję zacząć od przygotowania sobie procedurek dla każdego z rodzaju działań (zdejmij ze stosu, zdejmij ze stosu, wykonaj na obu liczbach co trzeba, wrzuć na stos) – będzie dużo łatwiej, szczególnie że wszystkie będą podobne, a potem można je sobie będzie wywoływać albo kopiować, co kto lubi.

Na plusika oczywiście publiczna prezentacja rozwiązania. Na końcu znajdziecie również zadanie na 2 pkt z terminem wykonania na po świętach.

Zadania podstawowe

1. Napisz program obliczający wzór (skorzystaj z notacji postfiksowej): $a * b + c$
2. Napisz program obliczający wzór (skorzystaj z notacji postfiksowej): $2a + 2b - 2c$
3. Napisz program obliczający wzór (skorzystaj z notacji postfiksowej): $\frac{a}{b} + c$
4. Napisz program obliczający wzór (skorzystaj z notacji postfiksowej): $\frac{a}{b+c}$
5. Napisz program obliczający wzór (skorzystaj z notacji postfiksowej): $a\frac{b}{c}$
6. Napisz program obliczający wzór (skorzystaj z notacji postfiksowej): $a^2 + 2b + c$
7. Napisz program obliczający wzór (skorzystaj z notacji postfiksowej): $2a\frac{b}{2c}$

Zadanie dodatkowe

Napisz program, który w pamięci będzie miał zadeklarowany ciąg znaków w ONP i dla tego ciągu wyliczy wynik podanego działania. Program ma działać dla każdego prawidłowego ciągu z ONP, dowolnej długości. Dla uproszczenia przyjmijmy, że pracujemy tylko na liczbach dodatnich, z zakresu 1-9 oraz z góry znana jest długość ciągu.