Procesor stosowy

Państwo Gedanum postanowiło podbić kosmos. W pierwszym etapie planuje wysłać szereg sond, które będą poszukiwać planet nadających się do kolonizacji. Sondy muszą być inteligentne i energooszczędne, do sterowania nimi potrzebny jest więc nowy procesor. Zaprojektowanie go zlecono komisji Ekspertów Technologii Interstelarnych. Po wielu godzinach ciężkiej pracy i licznych przebłyskach geniuszu powstał Stosowy Translator Obiektów Symbolicznych, model A15D. Rozpoczęcie masowej produkcji wymaga jeszcze wiele pracy, jednak by nie tracić czasu, zlecono Tobie napisanie emulatora tego procesora. Dzięki niemu tworzenie systemów sterowania sondami będzie mogło rozpocząć się znacznie szybciej.

Procesor posiada pamięć programu, wskaźnik instrukcji oraz stos. Pamięć programu przechowuje program. Program jest sekwencją instrukcji, a każda instrukcja jest jednym znakiem. Wskaźnik instrukcji przechowuje numer instrukcji, która będzie wykonana następna i zwiększa się o jeden po jej wykonaniu (w przypadku większości instrukcji). Procesor kończy działanie, gdy wskaźnik instrukcji wskaże poza ostatnią instrukcje w programie. Pierwsza instrukcja ma numer 0.

Stos przechowuje dane do obliczeń. Elementami stosu są listy zawierające zero lub więcej znaków (cyfr, liter i innych symboli). Jeżeli lista zawiera same cyfry i co najwyżej jeden znak '-' (minus) na końcu, mówimy, że lista taka zawiera liczbę. Najmłodsza cyfra liczby jest przechowywana na pierwszym miejscu listy, zatem liczba -1234 jest przechowywana w postaci listy 4321-.

Można założyć, że wszystkie programy będą poprawne, tzn. nie wystąpią sytuacje nie przewidziane w liście instrukcji.

Lista instrukcji procesora jest następująca:

'	apostrof	włóż na stos pustą listę
,	przecinek	zdejmij listę ze stosu
:	dwukropek	włóż na stos kopię listy z wierzchołka stosu
;	średnik	zamień miejscami listę na wierzchołku stosu i listę bezpośrednio pod nim
@	at	zdejmij ze stosu liczbę A, następnie włóż na stos kopię listy na A-tej pozycji na stosie (0 to wierzchołek stosu, 1 lista bezpośrednio pod nim itp.); program '0@ jest równoważny instrukcji :
.	kropka	wczytaj znak ze standardowego wejścia i dołącz go na początek listy na wierzchołku stosu
>	większe niż	wypisz na standardowe wyjście pierwszy znak z listy na wierzchołku stosu i zdejmij listę z wierzchołka stosu
!	wykrzyknik	negacja logiczna: jeżeli na wierzchołku stosu jest pusta lista lub lista zawierająca pojedyńczy znak '0', zastąp ją listą zawierającą znak '1', w przeciwnym wypadku zastąp listę z wierzchołka stosu listą zawierającą znak '0'
<	mniejsze niż	zdejmij ze stosu liczbę A, zdejmij ze stosu liczbę B, jeżeli B < A, umieść na stosie liczbę 1, w przeciwnym wypadku umieść na stosie liczbę 0
=	równe	zdejmij ze stosu liczbę A, zdejmij ze stosu liczbę B, jeżeli B = A, umieść na stosie liczbę 1, w przeciwnym wypadku umieść na stosie liczbę 0
~	tylda	umieść na stosie liczbę równą numerowi tej instrukcji (wartość wskaźnika instrukcji)

?	znak zapytania	skok warunkowy: zdejmij ze stosu liczbę T, zdejmij ze stosu listę W, jeżeli W nie jest pusta i nie zawiera wyłącznie znaku '0', wpisz do wskaźnika instrukcji liczbę T i nie zwiększaj wskaźnika instrukcji;
-	minus	negacja: jeżeli ostatnim znakiem listy na szczycie stosu jest '-' (minus), usuń go z listy; w przeciwnym wypadku dołącz '-' na koniec listy na szczycie stosu
^	daszek	wartość bezwzględna: jeżeli ostatnim znakiem listy na szczycie stosu '-' (minus) usuń go z listy
$	dolar	podział: odłącz pierwszy znak z listy na szczycie stosu i włóż go na stos
#	hasz	zdejmij ze stosu listę A; dołącz A na koniec listy na szczycie stosu
+	plus	zdejmij ze stosu liczbę A, zdejmij ze stosu liczbę B, włóż na stos liczbę A + B
&	ampersand	wypisz na standardowe wyjście zawartość stosu w formacie: n: lista na n-tej pozycji na stosie ... 1: lista na drugim miejscu na stosie 0: lista na wierzchołku stosu lista powinna być przedstawiona jako ciąg kolejnych znaków z listy (zaczynając od pierwszego)
]		zdejmij ze stosu liczbę A, umieść na stosie znak o numerze A w ASCII
[		zdejmij ze stosu listę A, umieść na stosie liczbę równą numerowi ASCII pierwszego znaku z listy A
pozostałe znaki		dołącz ten znak na początek listy na szczycie stosu

Wejście

Pierwsza linia wejścia zawiera program do wykonania. W drugiej linii podane będą znaki, które będą podawane na standardowe wejście programu. Ani program ani wejście programu nie będzie zawierało białych znaków. Ilość instrukcji nie przekroczy 20000. Liczba znaków podanych na standardowe wejście nie przekroczy 20000.

Wyjście

Na wyjście należy wypisać wszystko to, co procesor wypisałby na standardowe wyjście.

Zasady i punktacja

Program nie powinien korzystać z STL (poza obsługą wejścia i wyjścia) a także z tablic (tablicy można użyć wyłącznie do przechowywania programu i danych wejściowych). Poza wczytywaniem wejścia, pętli for/while/do...while można użyć co najwyżej raz (należy skorzystać z rekurencji).

Testy podzielone są na grupy, każda kolejna grupa rozszerza grupę poprzedzającą (nie można zaliczyć grupy np. 6 nie zaliczając grup 1, 2, 3, 4 i 5).

Uwaga: testy są tak skonstruowane, aby przy rozsądnych parametrach do funkcji stosu wystarczyło. Jeżeli pod Visual Studio wstępuje stack overflow, nie znaczy to jeszcze, że na STOSie tak będzie (tu rozmiar stosu jest ustawiony na 10MB). Dostępny rozmiar stosu można w Visual Studio ustawić w opcjach projektu -> Linker -> System -> Stack Reserve Size (domyślnie jest 1MB).

Testy

1-3 (12.5%)
instrukcje ' , : ; @ & pozostałe znaki, na stosie przechowywane będą tylko liczby mieszczące się w zakresie int-a

4-6 (12.5%)
instrukcje ' , : ; @ & pozostałe znaki

7-9 (12.5%)
dodatkowo instrukcje . > - ^

10-12 (12.5%)
dodatkowo instrukcje [ ] $ #

13-16 (25%)
dodatkowo instrukcje < = ! ~ ?

17-19 (25%)
dodatkowo instrukcja +

Przykład

Wejście:
```
'123'-456&+&
```

Wyjście:
```
1: 321
0: 654-
0: 333-
```

Wejście:
```
'...&$&
123
```

Wyjście:
```
0: 321
1: 21
0: 3
```