|  |  |  |
| --- | --- | --- |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное автономное  образовательное учреждение высшего образования  «Пермский государственный национальный  исследовательский университет» | | |
|  | Институт компьютерных наук и технологий | |
| **ОТЧЁТ**  по индивидуальной работе №2  по дисциплине «Языки программирования»  Вариант 4 | | |
|  | | Работу выполнил  студент группы ПМИ-9,10-2023 1 курса  Румянцев В.И.  «16» июня 2024 г. |
| Работу проверил  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Фамилия И.О.  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2024 г. |
| Пермь 2024 | | |

СОДЕРЖАНИЕ

[Постановка задачи 3](#_Toc169573647)

[Алгоритм решения 5](#_Toc169573648)

[Тестирование 6](#_Toc169573649)

[Код программы 10](#_Toc169573650)

# Постановка задачи

Абстрактная вычислительная машина имеет один регистр и шесть инструкций:

LD A — Помещает операнд А в регистр

ST A — Помещает содержимое регистра в переменную с именем А

AD A — Прибавляет значение А к регистру. Результат остается в регистре.

SB A — Вычитает значение А из регистра. Результат остается в регистре.

ML A — Умножает содержимое регистра на значение переменной с именем A. Результат остается в регистре.

DV A — Делит содержимое регистра на значение А. Результат остается в регистре.

Дано выражение в постфиксной форме, состоящее из однобуквенных операндов и операций «+», «–», «\*», «/». Написать программу вывода инструкций вычислительной машины, необходимых для вычисления выражения. Результат вычисления должен оставаться в регистре, разрешается использовать обозначения Tn для временных переменных.

Пример: для выражения ABC\*+DE-/ перечень инструкций имеет вид

LD B

ML C

ST T1

LD A

AD T1

ST T1

LD D

SB E

ST T2

LD T1

DV T2

Замечание: Постфиксной формой записи выражения a b называется запись, в которой знак операции размещен за операндами a b.

Примеры:

a - b → a b –

a \* b + c → a b \* c +

a \* (b + c) → a b c + \*

a + b / c / d \* e → a b c / d / e \* +

# Алгоритм решения

1. Определил класс `Node` для создания узла связанного списка. Узел содержит данные и ссылку на следующий узел.

2. Определил класс `LinkedList` для создания односвязного списка. Список содержит ссылку на первый узел (голову) и методы для добавления и удаления узлов, а также для преобразования списка в строку и определения его длины.

3. Определил класс `Stack` для создания стека. Стек содержит связанный список для хранения данных, число `n` для временных переменных `Tn` и связанный список `free\_names` для хранения освободившихся временных переменных. Кроме того, класс `Stack` содержит метод `get\_name\_of\_temp\_var` для получения имени для временной переменной `Tn` и метод `generate\_instructions` для добавления нового элемента в стек и генерации инструкций вычислительной машины.

4. Определил функцию `main`, которая принимает на вход выражение в постфиксной форме. Функция создает стек, проходит по каждому элементу входного выражения и добавляет его в стек с помощью метода `generate\_instructions`. Затем функция проверяет, остались ли элементы в стеке, и формирует итоговый ответ, содержащий инструкции вычислительной машины.

5. В основном блоке программы получаем ввод от пользователя, запускаем функцию `main` и обрабатываем исключения, которые могут возникнуть во время выполнения программы.

6. Добавил комментарии к коду для лучшего понимания работы программы.

7. Проверил работу программы на различных входных данных.

# Тестирование

1. Запрос: «ABC\*+DE-/»

Ответ: «

Результат:

LD B

ML C

ST T1

LD A

AD T1

ST T1

LD D

SB E

ST T2

LD T1

DV T2

»

1. Запрос: «AB+CD+-EF\*GH/+\*»

Ответ: «

Результат:

LD A

AD B

ST T1

LD C

AD D

ST T2

LD T1

SB T2

ST T1

LD E

ML F

ST T2

LD G

DV H

ST T3

LD T2

AD T3

ST T2

LD T1

ML T2

»

1. Запрос: «abc/d/e\*+»

Ответ: «

Результат:

LD b

DV c

ST T1

LD T1

DV d

ST T1

LD T1

ML e

ST T1

LD a

AD T1

»

1. Запрос: «»

Ответ: «

Результат отсутствует!

Пожалуйста, перепроверьте корректность вашего выражения и запросите новый результат.

»

1. Запрос: «ABC-»

Ответ: «

Программа окончила работу с лишним(и) операндом(ами) в регистре: "A".

Это связано с излишним их количеством на входе (например, было подано выражение "ABC+").

Совет: В случае, если вы не уверены в корректности вашего запроса - перепроверьте выражение, которое было подано программе на вход и запросите новый результат!

Результат:

LD B

SB C

»

1. Запрос: «A/»

Ответ: «

Во время работы программы произошла ошибка из-за нехватки переменных для работы.

Такая ошибка происходит, например, когда на вход программе подано такое выражение: "A+", вместо такого "AB+".

Перепроверьте ваше выражение и введите его корректно.

»

1. Запрос: «JVSNVFKS»

Ответ: «

Результат отсутствует!

Пожалуйста, перепроверьте корректность вашего выражения и запросите новый результат.

»

1. Запрос: «АППААП++++++»

Ответ: «

Во время работы программы произошла ошибка из-за нехватки переменных для работы.

Такая ошибка происходит, например, когда на вход программе подано такое выражение: "A+", вместо такого "AB+".

Перепроверьте ваше выражение и введите его корректно.

»

1. Запрос: «Абапре+++++»

Ответ: «

Результат:

LD р

AD е

ST T1

LD п

AD T1

ST T1

LD а

AD T1

ST T1

LD б

AD T1

ST T1

LD А

AD T1»

# Код программы

# Вариант 4. Румянцев Владимир ПМИ-10.

class Node:

def \_\_init\_\_(self, data=None, next=None):

self.data = data

self.next = next

def \_\_str\_\_(self):

return str(self.data)

class LinkedList: # Односвязный список

def \_\_init\_\_(self):

self.head = None

def \_\_str\_\_(self) -> str: # Красивое написание всего списка (все элементы через запятую в квадратных скобках)

current = self.head

line = ''

while current:

line += f'{current.data}, '

current = current.next

return f'[{line[:-2]}]'

def \_\_len\_\_(self) -> int: # Узнать количество элементов в списке

current = self.head

count = 0

while current:

current = current.next

count += 1

return count

def push(self, new\_data): # Добавить в начало списка новый элемент

new\_node = Node(new\_data)

new\_node.next = self.head

self.head = new\_node

def add(self, new\_data): # Добавить в конец списка новый элемент

if not self.head:

self.head = Node(new\_data)

return

current = self.head

while current.next:

current = current.next

current.next = Node(new\_data)

def pop(self, index: int): # Удалить элемент по индексу, вернуть значение удаляемого элемента

if (index > len(self) - 1) or (index < 0): # Если такого индекса в списке нет - вернуть None

return None

if index == 0: # Если удаляется первый элемент - нужно переопределить начало списка

a = self.head.data

self.head = self.head.next

return a

current = self.head

position = 1

while current.next and (position < index): # Находим элемент перед удаляемым

current = current.next

position += 1

a = current.next.data # Записываем значение удаляемого элемента (current - элемент перед удаляемым)

current.next = current.next.next # Переопределяем ссылки элемента перед удаляемым

return a # Возвращаем значение удаляемого элемента

class Stack: # Стек - объект для хранения всей необходимой информации

def \_\_init\_\_(self):

self.list = LinkedList() # Тут будут храниться операнды

self.n = 1 # Число n для временных переменных Tn

self.free\_names = LinkedList() # Пополняется освободившимися именами Tn

def get\_name\_of\_temp\_var(self, var1, var2): # Функция получения имени для временной переменной Tn

if (len(var1) > 1) and (var1[0] == 'T'): # Если первый операнд в формате Tn, ...

self.free\_names.add(var1) # добавить его в список с освободившимися именами Tn

if (len(var2) > 1) and (var2[0] == 'T'): # Если второй операнд в формате Tn, ...

self.free\_names.add(var2) # добавить его в список с освободившимися именами Tn

if self.free\_names.head: # Если в списке освободившихся Tn что-то есть,

return self.free\_names.pop(0) # то вернуть первый элемент

temp\_var = f'T{self.n}' # Если освободившихся имён нет, то создать и вернуть новое имя,

self.n += 1 # исходя из "self.n". Так-же увеличить "self.n" на 1

return temp\_var

def generate\_instructions(self, new\_data=None): # Генератор инструкций для новопришедшего элемента

if not (new\_data in '\*/+-'): # Если это операнд, то просто добавить его в стек

self.list.push(new\_data)

return '' # Возвращаем пустую строку

else: # Если это операция...

if new\_data == '+':

operation = 'AD'

elif new\_data == '-':

operation = 'SB'

elif new\_data == '\*':

operation = 'ML'

else:

operation = 'DV'

variable\_2 = self.list.pop(0) # Получить последние 2 операнда (последний пришедший в начале списка)

variable\_1 = self.list.pop(0)

if variable\_1 is None: # В стеке не хватает операндов, вызвать ошибку и окончить работу

raise Exception(f'Во время работы программы произошла ошибка из-за нехватки переменных для работы.\n'

f'Такая ошибка происходит, например, когда на вход программе подано такое выражение: '

f'"A+", вместо такого "AB+".\nПерепроверьте ваше выражение и введите его корректно.')

temp\_var = self.get\_name\_of\_temp\_var(variable\_1, variable\_2) # Получить имя для временной переменной

line = f'LD {variable\_1}\n{operation} {variable\_2}\nST {temp\_var}\n' # Перечень инструкций

self.list.push(temp\_var)

return line # Возвращаем перечень инструкций

def main(input\_data): # Главная функция, на входе получает выражение в постфиксной форме

stack = Stack()

line = ''

for i in input\_data: # Проходимся по каждому элементу в строке входных данных

line += stack.generate\_instructions(i) # В line дописываем результат функции (конкретные инструкции)

res\_line = '' # Создание переменной для итогового ответа пользователю

try: # Проверка на то, остались ли элементы в стеке,

current = stack.list.head.next # в норме останется только один элемент

if current: # Если элементов в стеке осталось больше одного, ...

res\_line = f'\nПрограмма окончила работу с лишним(и) операндом(ами) в регистре: '

while current: # ... то сообщим об этом в итоговом ответе

res\_line += f'"{current.data}", '

current = current.next

res\_line = (f'{res\_line[:-2]}.\n'

f'Это связано с излишним их количеством на входе (например, было подано выражение "ABC+").\n'

f'Совет: В случае, если вы не уверены в корректности вашего запроса - перепроверьте выражение, '

f'которое было подано программе на вход и запросите новый результат!\n')

except AttributeError: # Ошибка произошла из-за отсутствия в стеке даже одного элемента,

raise Exception('Результат отсутствует!\n' # такое происходит при пустых входных данных

'Пожалуйста, перепроверьте корректность вашего выражения и запросите новый результат.')

for i in range(len(line) - 2, -1, -1): # Т.к. согласно условию результат вычислений остаётся в регистре,

if (line[i] == 'S') and (line[i + 1] == 'T'): # нужно избавиться от последней инструкции ST

res\_line += f'\nРезультат:\n{line[:i].strip()}' # В переменную с конечным ответом записываем результат

print(res\_line) # Сообщить пользователю итог (перечень инструкций)

return # Программа прекращает цикл работы

print('Результат отсутствует!\n' # Некорректный ввод данных от пользователя, закончить цикл работы

'Пожалуйста, перепроверьте корректность вашего выражения и запросите новый результат.')

return

if \_\_name\_\_ == '\_\_main\_\_':

user\_input = input('\nВведите выражение в постфиксной форме (пример: ABC\*+): ') # Ввод пользователя

while True:

try:

main(user\_input) # Начало работы с данными от пользователя

except Exception as ex: # В случае ошибки, о ней сообщаем пользователю

print(ex)

finally: # Вне зависимости от результата, просим новое выражение

user\_input = input('\n----------------------------------------------------------------'

'----------------------------------------------------------------\n'

'\nВведите новое выражение: ')