МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического занятия 8

Тема: Применение стека и очереди при преобразовании арифметических выражений в постфиксную, префиксную нотации и вычисление значений выражений

Дисциплина: Структуры и алгоритмы обработки данных

Выполнил студент Хвостов В. В.

Группа ИКБО-01-20

Москва 2021

Содержание

Задание 1

Вариант 2.

1.1. Упражнение 1

Условие: Провести преобразование инфиксной записи выражения в постфиксную нотацию, расписывая процесс по шагам S=x-(y*a/b-(z+d*e)+c)/f

Алгоритм преобразования отображен в таблице 1.

Таблица 1 - Упражнение 1 задания 1

Токен	Действие	Вывод	Стек	Комментарий
x	Добавить токен в вывод	x		
-	Положить токен в стек	x	-	
(	Положить токен в стек	x	(-	
y	Добавить токен в вывод	xy	(-	
*	Положить токен в стек	xy	*(-	
a	Добавить токен в вывод	xya	*(-	
/	Положить токен в стек	xya	/*(-	
b	Добавить токен в вывод	xyab	/*(-	
-	Добавлять токен в вывод пока он не равен '('	xyab/*	(-	
	Положить токен в стек	xyab/*	-(-	
(	Положить токен в стек	xyab/*	(-(-	
z	Добавить токен в вывод	xyab/*z	(-(-	
+	Положить токен в стек	xyab/*z	+(-(-	
d	Добавить токен в вывод	xyab/*zd	+(-(-	
*	Положить токен в стек	xyab/*zd	*+(-(-	
e	Добавить токен в вывод	xyab/*zde	*+(-(-	
)	Добавить токен в вывод из стека	xyab/*zde*+	(-(-	Повторять пока не найдена (
	Удалить из стека токен	xyab/*zde*+	-(-	Убираем скобки
+	Положить токен в стек	xyab/*zde*+	+-(-	
c	Добавить токен в вывод	xyab/*zde*+c	+-(-	

)	Добавить токен в вывод из стека	xyab/*zde*+c+-	(-	Повторять пока не найдена (
	Удалить из стека токен	xyab/*zde*+c+-	-	Убираем скобки
/	Положить токен в стек	xyab/*zde*+c+-	/-	
f	Добавить токен в вывод	xyab/*zde*+c+-f	/-	
Конец	Добавляем оставшиеся токены из стека	xyab/*zde*+c+-f/-		

Результат: xyab/*zde*+c+-f/-.

1.2. Упражнение 2

Представить постфиксную нотацию выражний:

1. (((a+b*c)/d+(x*y-z/k)+m)*n)
2. a+b/c+d+e-f/m*k

Алгоритм преобразования показан на таблицах 2 и 3.

Таблица 2 - Упражнение 2.1 задания 1

Токен	Вывод	Стек
(		(
(		((
(		(((
a	a	(((
+	a	+(((
b	ab	+(((
*	ab	*+(((
c	abc	*+(((
)	abc*+	((
/	abc*+	/((
d	abc*+d	/((
+	abc*+d/	+((
(	abc*+d/	(+((
x	abc*+d/x	(+((
*	abc*+d/x	*(+((

y	abc*+d/xy	*(+((
-	abc*+d/xy*	-(+((
z	abc*+d/xy*z	-(+((
/	abc*+d/xy*z	/-(+((
k	abc*+d/xy*zk	/-(+((
)	abc*+d/xy*zk/-	+((
+	abc*+d/xy*zk/-	++((
m	abc*+d/xy*zk/-m	++((
)	abc*+d/xy*zk/-m++	(
*	abc*+d/xy*zk/-m++	*(
n	abc*+d/xy*zk/-m++n	*(
)	abc*+d/xy*zk/-m++n*	

Результат: abc*+d/xy*zk/-m++n*.

Таблица 3 - Упражнение 2.2 задания 1

Токен	Вывод	Стек
a	a	
+	a	+
b	ab	+
/	ab	/+
c	abc	/+
+	abc/	++
d	abc/d	++
+	abc/d	+++
e	abc/de	+++
-	abc/de	-+++
f	abc/def	-+++
/	abc/def	/-+++
m	abc/defm	/-+++
*	abc/defm	*/-+++
k	abc/defmk	*/-+++
Конец	abc/defmk*/-+++	

Результат: abc/defmk*/-+++.

1.3. Упражнение 3

Представить префиксную нотацию выражний:

1. (((a+b*c)/d+(x*y-z/k)+m)*n)
2. a+b/c+d+e-f/m*k

Для преобразования в префиксную форму перевернем строки, а затем результат, полученный в результате преобразований, еще раз перевернем, что даст нам необходимый результат:

1. (n*(m+(k/z-y*x)+d/(c*b+a)))
2. k*m/f-e+d+c/b+a

Алгоритм преобразования показан на таблицах 4 и 5.

Таблица 4 - Упражнение 3.1 задания 1

Токен	Вывод	Стек
(		(
n	n	(
*	n	*(
(	n	(*(
m	nm	(*(
+	nm	+(*(
(	nm	(+(*(
k	nmk	(+(*(
/	nmk	/(+(*(
z	nmkz	/(+(*(
-	nmkz/	-(+(*(
y	nmkz/y	-(+(*(
*	nmkz/y	*-(+(*(
x	nmkz/yx	*-(+(*(
)	nmkz/yx*-	+(*(

+	nmkz/yx*-	++(*(
d	nmkz/yx*-d	++(*(
/	nmkz/yx*-d	/++(*(
(	nmkz/yx*-d	(/++(*(
c	nmkz/yx*-dc	(/++(*(
*	nmkz/yx*-dc	*(/++(*(
b	nmkz/yx*-dcb	*(/++(*(
+	nmkz/yx*-dcb*	+(/++(*(
a	nmkz/yx*-dcb*a	+(/++(*(
)	nmkz/yx*-dcb*a+	/++(*(
)	nmkz/yx*-dcb*a+/++	*(
)	nmkz/yx*-dcb*a+/++*	

Результат: *++/+a*bcd-*xy/zkmn.

Таблица 5 - Упражнение 3.2 задания 1

Токен	Вывод	Стек
k	k	
*	k	*
m	km	*
/	km	/*
f	kmf	/*
-	kmf/*	-
e	kmf/*e	-
+	kmf/*e	+-
d	kmf/*ed	+-
+	kmf/*ed	++-
c	kmf/*edc	++-
/	kmf/*edc	/++-
b	kmf/*edcb	/++-
+	kmf/*edcb/	+++-
a	kmf/*edcb/a	+++-
Конец	kmf/*edcb/a+++-	

Результат: -+++a/bcde*/fmk.

1.4. Упражнение 4

Провести вычисление значения выражения в префиксной форме, расписывая процесс по шагам +7/-9 3*2 5

Перевернем запись для нашего вычисления: 5 2*3 9-/7+ Процесс вычисления значения представленного выражения с использованием стека приведен в таблице 6.

Таблица 6 - Упражнение 4 задания 1

Токен	Стек	Результат
5	5	
2	2 5	
*		2 * 5 = 10
3	3 10	
9	9 3 10	
-	10	9 - 3 = 6
/		6 / 10 = 0
7	7 0	
+		7 + 0 = 7
Конец	7	

Результат = 7.

Задание 2

2.1. Задача 1

2.1.1. Условие задачи

Реализовать операции над очередью: втолкнуть элемент в очередь, вытолкнуть элемент из очереди, вернуть значение элемента в вершине очереди, сделать очередь пустой, определить пуста ли очередь. Рассмотреть два варианта реализации очереди: на массиве (или строке); на однонаправленном списке.

- Создать класс или просто заголовочный файл с функциями.

- Применить операции для вычисления значения выражения п.1.4 варианта 2.

2.1.2. Постановка задачи

Выбрать структуру данных для реализации очереди и описать функции работы с очередью, реализовать функцию вычисления значения выражения, записанного в префиксной форме.

2.1.3. Выбор структуры данных

В качестве структуры данных для стека рациональней использовать односвязный список с указателями на начало и конец списка т.к. данная структура имеет наименьшую асимптотическую сложность вставки и удаления элемента - $O(1)$. Информационной частью линейного списка является строковое значение (в общем случае - шаблон), хранящее токены из заданного выражения и результаты промежуточных подсчетов.

2.1.4. Код реализации структуры данных

```
#include <algorithm>
#include <iostream>
#include <locale>
#include <stack>
#include <string>
#include <utility>
#include <vector>

template<typename T>
class MyQueue final {
public:
  MyQueue() = default;
  ~MyQueue() {
      this->clear();
  }
  void push(const T& str) {
     Node* node = new Node();
     node->data = str;
     node->next = nullptr;
     if (begin == nullptr) {
        begin = (end = node);
     } else {
```

```
        end->next = node;
        end = node;
    }
    ++_size;
}
T pop() {
    if (begin == nullptr) {
        throw std::runtime_error("Pop on empty queue");
    }
    Node* node = begin;
    begin = begin->next;
    T str = node->data;
    delete node;
    --_size;
    return str;
}
T front() const {
    return begin->data;
}
T back() const {
    return end->data;
}
bool empty() const {
    return begin == nullptr;
}
void clear() {
    for (; begin != nullptr;) {
        Node* node = new Node();
        node = begin;
        begin=begin->next;
        delete node;
    }
}
T& operator[](size_t idx) {
    Node* node = begin;
    for (size_t i = 0; node->next != nullptr && i < idx; ++i, node=node->next);
    return node->data;
}
void print() const {
    for (Node* node = begin; node != nullptr; node=node->next) {
        std::cout << node->data << '\n';
    }
}
size_t size() const {
    return _size;
}
private:
struct Node {
```

```
    T data;
    Node* next;
  };
  Node* begin = nullptr;
  Node* end = nullptr; // not after end!
  size_t _size = 0;
};
```

2.1.5. Код вычисления выражений

```
bool isdigit(const std::string& str) {
  for (char c: str) {
    if (!std::isdigit(c)) {
      return false;
    }
  }
  return true;
}

int calculatePrefixQueue(MyQueue<std::string>& queue) {
  if (queue.size() == 1) {
    return std::stoi(queue.pop());
  }
  MyQueue<std::string> rec_queue;
  size_t old_size = queue.size();
  while (queue.size() > 0) {
    if (!isdigit(queue[0]) && isdigit(queue[1]) && isdigit(queue[2])) {
      char op = queue.pop().at(0);
      int o1 = std::stoi(queue.pop());
      int o2 = std::stoi(queue.pop());
      switch(op) {
        case '+':
          rec_queue.push(std::to_string(o1 + o2));
          break;
        case '-':
          rec_queue.push(std::to_string(o1 - o2));
          break;
        case '*':
          rec_queue.push(std::to_string(o1 * o2));
          break;
        case '/':
          rec_queue.push(std::to_string(o1 / o2));
          break;
      }
    } else {
      std::string element = queue.pop();
      rec_queue.push(element);
    }
  }
```

```
    if ( old_size > rec_queue . size ()) {
        return calculatePrefixQueue ( rec_queue );
    } else {
        throw std :: runtime_error ("Wtf bro");
    }
}
```

2.1.6. Код основной программы

```
#include <iostream>
#include <string>
#include "parsing .cpp"

int main() {
    MyQueue<std:: string > queue;
    std :: string expression ;
    std :: cout << "Enter an expression in prefix form: ";
    std :: cin >> expression ;
    for (char c: expression ) {
        std :: string str (1, c);
        queue . push( str );
    }
    std :: cout << "Value of the expression = " << calculatePrefixQueue (queue) << '\n';
    return 0;
}
```

2.1.7. Результаты тестирования

Таблица 7 - Тестирование задачи 1

Номер теста	Входные данные	Ожидаемый результат	Результат выполнения программы
1	Выражение: +7/-93*25	7	Enter an expression in prefix form: +7/-93*25 Value of the expression = 7
2	Выражение: +25+/84	9	Enter an expression in prefix form: ++25/84 Value of the expression = 9
3	Выражение: *+54+46	90	Enter an expression in prefix form: *+54+46 Value of the expression = 90

2.2. Задача 2

2.2.1. Условие задачи

Разработать функцию(ии) преобразования инфиксной формы скобочного арифметического выражения в префиксную форму

2.2.2. Постановка задачи

Реализовать функцию преобразования выражения из инфиксной формы в префиксную и проверить с результатами пункта 1.3.

2.2.3. Описание подхода решения задачи

Для этой функции лучше всего использовать алгоритм сортировочной станции для вывода в виде обратной польской нотации (постфиксной) перевернутого исходного выражения, а затем переворачивание результата для получения необходимой польской нотации (префиксной).

2.2.4. Алгоритм на псевдокоде и описание всех используемых переменных

Описание переменных: expression - выражение, которое нужно обработать; result - результат работы алгоритма; operators - стек, хранящий в себе все, что не является числами или символами (т.е. операторы); token - каждый символ в заданном выражении.

Алгоритм 1 Алгоритм преобразования из инфиксной в префисную форму

```
function convertFromInfixToPrefix(expression)
    result ← expression
    reverse(result)
    reverseParentheses(result)              ▷ Заменяет все '(' на ')' и наоборот
    result ← convertFromInfixToPostfix(result)
    reverse(result)
    return result
end function
```

Алгоритм 2 Алгоритм преобразования из инфиксной в постфиксную форму

```
function convertFromInfixToPostfix(expression)
    Let's operators - stack of the characters
    result ← ""
    for every token in a expression do
        if token is a number or character then
            operators.push(token)
        else if token is an operation then
            while operators is not empty and precidence of the top operator in operators greater than precidence of token and the top operator in operators != '(' do
                result ← result + the top operator in operators
                operators.pop()
            end while
            operators.push(token)
        else if token == '(' then
            operators.push(token)
        else if token == ')' then
            while the top operator in operators != '(' do
                result ← result + the top operator in operators
                operators.pop()
                if the top operator == '(' then
                    operators.pop()
                end if
            end while
        end if
    end for
    while operators is not empty do
        result ← result + the top operator in operators
        operators.pop()
    end while
    return result
end function
```

2.2.5. Код программы

Код алгоритма:

```
bool isOperator(char c) {
  return c == '+' || c == '-' || c == '*' || c == '/';
}

int getPrecidence(char c) {
  if (c == '*' || c == '/') {
    return 2;
  } else if (c == '-' || c == '+') {
    return 1;
  } else {
    return 0;
  }
}

// Shunting-yard algorithm
std::string convertFromInfixToPostfix(const std::string& expression) {
  std::stack<char> operators;
  std::string result = "";
  for (char token: expression) {
    if (std::isdigit(token) || std::isalpha(token)) {
      result += token;
    } else if (isOperator(token)) {
      while (!operators.empty()
          && getPrecidence(operators.top()) > getPrecidence(token)
          && operators.top() != '(') {
        result += operators.top();
        operators.pop();
      }
      operators.push(token);
    } else if (token == '(') {
      operators.push(token);
    } else if (token == ')') {
      while (operators.top() != '(') {
        result += operators.top();
        operators.pop();
      }
      if (operators.top() == '(') {
        operators.pop();
      }
    }
  }
  while (!operators.empty()) {
    result += operators.top();
    operators.pop();
  }
```

```
    return result;
}

void reverseParentheses(std::string& expression) {
    for (char& c: expression) {
        if (c == '(') {
            c = ')';
        } else if (c == ')') {
            c = '(';
        }
    }
}


std::string convertFromInfixToPrefix(const std::string& expression) {
    std::string result = expression;
    std::reverse(result.begin(), result.end());
```

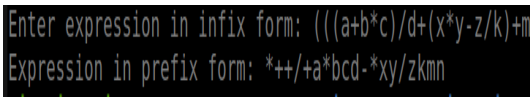
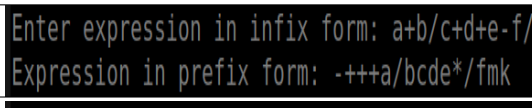
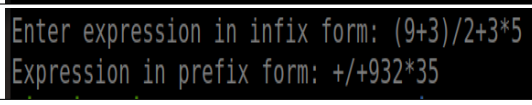
Код main:

```
#include <iostream>
#include <string>
#include "parsing.cpp"

int main() {
    std::cout << "Enter expression in infix form: ";
    std::string expression;
    std::cin >> expression;
    std::cout << "Expression in prefix form: "
        << convertFromInfixToPrefix(expression) << '\n';
    return 0;
}
```

2.2.6. Результаты тестирования

Таблица 8 - Тестирование задачи 2

Номер теста	Входные данные	Ожидаемый результат	Результат выполнения программы
1	Выражение: (((a+b*c)/d+(x*y-z/k)+m)*n)	*++/+a*bcd-*xy/zkmn	
2	Выражение: a+b/c+d+e-f/m*k	-+++a/bcde/fmk	
3	Выражение: (9+3)/2+3*5	+/+932*35	

2.3. Задача 3

2.3.1. Условие задачи

Дан текст, сформированный по правилу:

<текст>::=<пусто>|<элемент><текст>

<элемент>::=<буква>|(<текст>).

Требуется для каждой пары соответствующих открывающей и закрывающей скобок вывести номера их позиций в тексте, упорядочив пары номеров в порядке возрастания номеров позиций:

1. закрывающих скобок
2. открывающих скобок

Например, для текста A+(45-A(X)*(B-C)) должно быть выведено:

1. 8 10; 12 16; 3 17;
2. 3 17; 8 10; 12 16;

2.3.2. Постановка задачи

Реализовать функцию вывода номеров пар скобок в тексте, упорядочив их порядке возрастания либо закрывающих, либо открывающих скобок (по выбору пользователя).

2.3.3. Описание подхода решения задачи

Для решения задачи нам потребуется стек, в который мы будем класть индексы встречающихся открывающих ("левых") скобок. При встрече закрывающей ("правой") скобки, мы будем класть пару текущего индекса правой скобки и индекс на вершине стека в массив из пар индексов левых и правых скобок. Затем, по решению пользователя, мы сортируем массив быстрой сортировкой либо по левой, либо по правой скобке.

2.3.4. Алгоритм на псевдокоде и описание всех используемых переменных

Описание переменных: expression - выражение, которое нужно обработать; order - порядок, в котором происходит вывод пар индексов скобок. Остальные переменные описаны в комментариях псевдокода.

Алгоритм 3 Алгоритм функции вывода номеров пар скобок в тексте, упорядочив их по выбору пользователя

```
procedure outputGroupOfParanthesesByOrder(expression, order)
    Let's left_par_ids - stack of integers          ▷ стек из индексов левых скобок
    Let's v - vector of pairs of two integers       ▷ вектор из пар индексов скобок
    for i = 0 to the size of expression do
        if expression[i] == '(' then
            left_par_ids.push(i+1)
        else if expression[i] == ')' then
            left_id ← the top of left_par_ids                ▷ индекс левой скобки
            left_par_ids.pop()
            right_id ← i + 1                                 ▷ индекс правой скобки
            v.push_back(pair of left_id and right_id)
        end if
    end for
    if sort by right paranthesis then
        v is sorted by right paranthesis
    else
        v is sorted by left paranthesis
    end if
    for each element in v do
        Print to the screen pair of ids with ';' at the end
    end for
    Print end line
end procedure
```

2.3.5. Код программы

Код алгоритма:

```
enum class Order { LeftParenthesis , RightParenthesis };
void outputGroupOfParenthesesByOrder (const std :: string & expression ,
    Order order ) {
  std :: stack <int> left_par_ids ;
  using parIds = std :: pair <int , int >;
  std :: vector < parIds > v;
  for ( size_t i = 0; i < expression . size (); ++i) {
    if ( expression [i] == '(') {
      left_par_ids .push(i+1);
    } else if ( expression [i] == ')') {
      int left_id = left_par_ids .top();
      left_par_ids .pop();
      int right_id = i+1;
      v. push_back ({ left_id , right_id });
    }
  }
  if ( order == Order :: RightParenthesis ) {
    std :: sort (v. begin (), v.end(), [] (const parIds & p1, const parIds & p2)
        { return p1. second < p2. second ; });
  } else {
  std :: sort (v. begin (), v.end(), [] (const parIds & p1, const parIds & p2)
        { return p1. first < p2. first ; });
  }
  for (const auto& elem: v) {
    std :: cout << elem. first << ' ' << elem. second << "; ";
  }
  std :: cout << '\n';
}
```

Код main:

```
#include <iostream >
#include < string >

#include " parsing .cpp"

int main() {
  std :: string expression ;
  std :: cout << " Enter an expression to parse its parantheses : ";
  std :: cin >> expression ;
  std :: cout << " Enter the number to set order in which parantheses "
    << " indexes will be print on the screen :\n"
    << "1 - opening parantheses , 2 - close one\n";
  int order_num ;
  std :: cin >> order_num ;
  switch (order_num ) {
```

```
    case 1:  outputGroupOfParenthesesByOrder ( expression ,
                 Order ::  LeftParenthesis ) ;
             break;
    case 2:  outputGroupOfParenthesesByOrder ( expression ,
                 Order ::  RightParenthesis ) ;
             break;
  }
  return 0;
}
```

2.3.6. Результаты тестирования

Таблица 9 - Тестирование задачи 3

Номер теста	Входные данные	Ожидаемый результат	Результат выполнения программы
1	Выражение: A+(45-A(X)*(B-C)), Порядок: 1	3 17; 8 10; 12 16;	Enter an expression to parse its parantheses: A+(45-A(X)*(B-C)) Enter the number to set order in which parantheses indexes will 1 - opening parantheses, 2 - close one 1 3 17; 8 10; 12 16;
2	Выражение: A+(45-A(X)*(B-C)), Порядок: 2	8 10; 12 16; 3 17	Enter an expression to parse its parantheses: A+(45-A(X)*(B-C)) Enter the number to set order in which parantheses indexes will 1 - opening parantheses, 2 - close one 2 8 10; 12 16; 3 17;
3	Выражение: (B+(45*C(E))), Порядок: 1	1 13; 4 12; 9 11;	Enter an expression to parse its parantheses: (B+(45*C(E))) Enter the number to set order in which parantheses indexes w 1 - opening parantheses, 2 - close one 1 1 13; 4 12; 9 11;

Выводы

В ходе практической работы был рассмотрен метод применения стека и очереди при преобразовании арифметических выражений в постфиксную, префиксную нотации и вычислении значений выражений; получены знания и навыки по реализации структуры стек и очередь и функций преобразований выражений. Также были полученые навыки по написанию алгоритма на псевдокоде. Каждая функция прошла тестирование успешно, что подтверждает правильную работу алгоритмов.

Список информационных источников

1. Thomas H. Cormen, Clifford Stein и другие: Introduction to Algorithms, 3rd Edition. Сентябрь 2009. The MIT Press.

2. N. Wirth: Algorithms and Data Structures. Август 2004. https://people.inf.ethz.ch/wirth/AD.pdf.

3. Shunting-yard algorithm // Wikipedia [Электронный ресурс]. URL: https://en.wikipedia.org/wiki/Shunting-yard_algorithm (Дата обращения: 05.05.2021)

4. Курс Algorithms, part 2 // Coursera [Электронный ресурс]. URL: https://www.coursera.org/learn/algorithms-part2 (Дата обращения: 05.05.2021)