**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №3**

**по дисциплине «Программирование»**

Тема: **«Польские нотации. Стек и очередь»**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 0324 |  | Жигалова Д.А. |
| Преподаватель |  | Глущенко А.Г |

Санкт-Петербург

2021

# **Цель работы**

Получение практических навыков работы со стеками и очередями; изучение обратной и прямой польской нотации; проведение сравнительного анализа этих структур данных.

# **Основные теоретические положения**

Структуры представляют собой группы связанных между собой, как правило, разнотипных переменных, объединенных в единый объект, в отличие от массива, все элементы которого однотипны. В языке C++ структура является видом класса и обладает всеми его свойствами.

Стек – это частный случай однонаправленного списка, добавление элементов в который и выборка из которого выполняется с одного конца, называемого вершиной стека. Другие операции со стеком не определены. При выборке элемент исключается из стека. Говорят, что стек реализует принцип обслуживания LIFO (последним пришел – первым ушел).

Основные операции над стеками:

1. чтение верхнего элемента;
2. добавление нового элемента;
3. удаление существующего элемента.

Графически его удобно изобразить в виде вертикального списка (рис. 1), например, стопки книг, где для того чтобы воспользоваться одной из них и не нарушить установленный порядок, нужно поднять все книги, которые лежат выше нее, а положить книгу можно лишь поверх всех остальных.

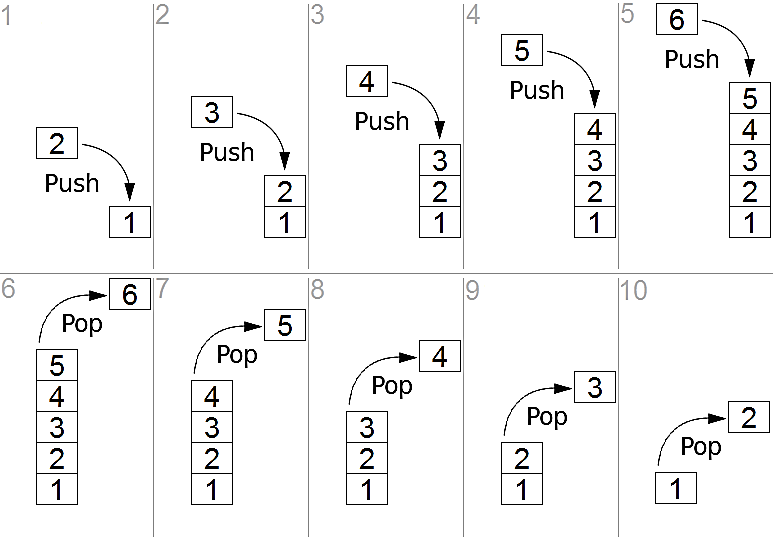


Рис. 1. Графическое представление стека

Стек чаще всего реализуется на основе обычных массивов, односвязных и двусвязных списков. В зависимости от конкретных условий выбирается одна из этих структур данных.

Очередь – частный случай однонаправленного списка, добавление элементов в который выполняется в один конец, а выборка – из другого конца. Другие операции с очередью не определены. При выборке элемент исключается из очереди. Говорят, что очередь реализует принцип обслуживания FIFO (первым пришел – первым ушел). В программировании очереди применяются при моделировании, диспетчеризации задач операционной системой, буферизованном вводе/выводе.

Графически ее удобно изобразить в виде вертикального списка (рис. 2), например, очередь в магазине, где для того чтобы дойти до кассы и не нарушить установленный порядок, нужно дождаться, пока все покупатели перед вами не приобретут товар. Разумеется, будут появляться новые покупатели, которые будут занимать свое место в очереди в ожидании покупки.

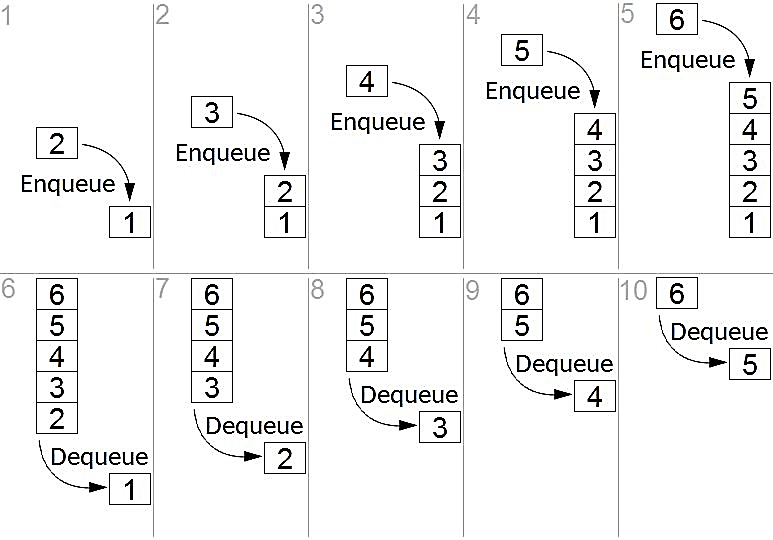


Рис. 2. Графическое представление очереди

Основные операции над очередями:

1. чтение первого элемента;
2. добавление нового элемента;
3. удаление существующего элемента.

Если для стека в момент добавления или удаления элемента допустимо задействование лишь его вершины, то касательно очереди эти две операции должны быть применены так, как это регламентировано в определении этой структуры данных, т. е. добавление – в конец, удаление – из начала.

Выделяют два способа программной реализации очереди. Первый основан на базе массива, второй – на базе указателей (связного списка). Первый способ – статический, так как очередь представляется в виде простого статического массива, второй – динамический.

Кольцевой буфер так же известен, как кольцевая очередь или циклический буфер и является распространенной формой очереди. Это популярный, легко реализуемый стандарт, и хотя он представлен в виде круга (рис. 3) в базовом коде он является линейным. Кольцевая очередь существует как массив фиксированной длины с двумя указателями: один представляет начало очереди, другой – хвост.

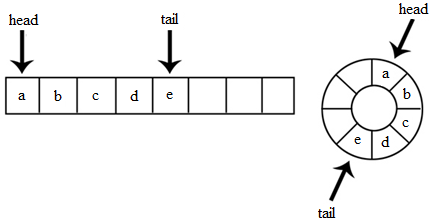


Рис. 3. Кольцевая очередь

Операции с очередями работают следующим образом.

1. Два указателя, называемые и , используются для отслеживания первого и последнего элементов в очереди.
2. При инициализации очереди значения и устанавливаются равными –1.
3. При добавлении элемента постепенно увеличивается значение индекса и помещается новый элемент в положение, на которое указывает .
4. При снятии очереди с элемента возвращается значение, на которое указывает , и постепенно увеличивается индекс .
5. Перед постановкой в очередь проверяется, заполнена ли очередь.
6. Перед снятием очереди проверяется, пуста ли очередь.
7. При инициализации первого элемента устанавливается значение в 0.
8. При удалении последнего элемента сбрасываются значения и в –1.

Недостатком метода является его фиксированный размер. Для очередей, где элементы должны быть добавлены и удалены в середине, а не только в начале и конце буфера, реализация в виде связанного списка является предпочтительным подходом.

Обратная польская запись (нотация) (рис. 4) – форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. Обратная польская запись имеет ряд преимуществ перед инфиксной записью при выражении алгебраических формул, одно из них то, что инфиксные операторы имеют приоритеты, которые произвольны и нежелательны.

|  |  |  |
| --- | --- | --- |
| Простое выражение | Прямая польская запись | Обратная польская запись |
| X + 3 \* Y  (X + 3) \* Y  1 + 2 | + X \* 3 Y  \* + X 3 Y  + 1 2 | X 3 Y \* +  X 3 + Y \*  1 2 + |

Рис. 4. Представление прямой и обратной польской записи

Обратная польская запись отлично подходит для вычисления выражений при помощи стека. Причем сам алгоритм достаточно прост. Необходимо просто прочитать обратную польскую запись слева направо. Если встречается операнд, то его нужно поместить в стек. Если встречается оператор, нужно выполнить заданную им операцию.

# **Постановка задачи**

Необходимо написать программу, которая выполняет следующее:

1. Реализует преобразование введенного выражения (если используются переменные, то пользователь должен их инициализировать). Ввод выражения должен быть реализовать двумя способами: с клавиатуры и с файла.

2. Реализует проверку на корректность простого выражения и выражения, записанного в прямой и обратной польских нотациях (на выбор пользователя). Ввод выражения должен быть реализовать двумя способами: с клавиатуры и с файла.

3. Реализует вычисления простого выражения и выражения, записанного в прямой и обратной польских нотациях (на выбор пользователя). Ввод выражения должен быть реализовать двумя способами: с клавиатуры и с файла.

4. Генерирует несколько (на выбор пользователя) вариантов проверочной работы по польской нотации (прямой и обратной). Задание и ответы к ним необходимо вывести в отдельные файлы (ответы должны быть максимально подробными).

Программа должна выводить и описывать все промежуточные действия.

**Выполнение работы**

Для решения поставленных задач была создана программа на языке программирования C++. Итоговый код программы представлен в приложении А, а результат работы в приложении B.

**Вывод**

Выполнив данную работу мы получили практический навык работы со стеками и очередями; изучили обратную и прямую польскую нотацию; провели сравнительный анализ этих структур данных.

# **ПРИЛОЖЕНИЕ А**

**ПОЛНЫЙ КОД ПРОГРАММЫ**

#include <iostream>

#include <cctype>

#include <stdlib.h>

#include <cstdio>

#include <stack>

#include <string>

using namespace std;

int priority(char o)

{

switch (o)

{

case '(': return 1;

case '+': case '-': return 2;

case '\*': case '/': return 3;

}

}

bool sign(char c)

{

return c == '+' || c == '-' || c == '\*' || c == '/';

}

bool num(char c)

{

return c >= '0' && c <= '9';

}

void RPN(char\* a, char\* out)

{

stack <char> S;

int i = 0, j = 0;

for (; a[i] != '\0'; ++i)

{

if (num(a[i]))

{

out[j] = a[i];

++j;

}

else

{

if (j != 0 && num(out[j - 1]))

{

out[j] = ' ';

++j;

}

if (sign(a[i]))

{

if (S.empty() || priority(S.top()) < priority(a[i]))

{

S.push(a[i]);

}

else

{

while (!S.empty() && (priority(S.top()) >= priority(a[i])))

{

out[j++] = S.top();

S.pop();

}

S.push(a[i]);

}

}

else

{

if (a[i] == '(')

{

S.push(a[i]);

}

else if (a[i] == ')')

{

if (S.empty() || S.top() == '(')

{

cout << "Input error!";

exit(1);

}

else

{

while (S.top() != '(')

{

out[j] = S.top();

S.pop();

j++;

}

}

S.pop();

}

else if (a[i] != ' ')

{

cout << "Initialize the variable " << a[i] << ": ";

cin >> a[i];

i--;

}

}

}

}

while (!S.empty())

{

if (S.top() == '(')

{

cout << "Input error!";

exit(1);

}

else

{

out[j] = S.top();

S.pop();

j++;

}

}

}

int calc(char\* out)

{

int j = 0, r1 = 0, r2 = 0;

stack <double> S;

stack <double> S1;

char\* pEnd = nullptr;

while (out[j] != '\0')

{

if (out[j] == '\_') {

++j;

continue;

}

if (out[j] >= '0' && out[j] <= '9')

{

long iNum = strtol(&out[j], &pEnd, 10);

S.push(iNum);

j += pEnd - &out[j];

}

else

{

if (sign(out[j]))

{

if (S.empty()) return false;

else {

r1 = S.top(); S.pop();

if (S.empty()) return false;

r2 = S.top(); S.pop();

switch (out[j])

{

case '+': S.push(r2 + r1); break;

case '-': S.push(r2 - r1); break;

case '\*': S.push(r2 \* r1); break;

case '/': S.push(r2 / r1); break;

}

}

}

++j;

}

}

return (S.top());

}

bool check(char\* a) {

bool right = true;

if (a[0] == '+' || a[0] == '\*' || a[0] == '/') {

right = false;

}

else if (a[strlen(a) - 1] == '+' || a[strlen(a) - 1] == '-' || a[strlen(a) - 1] == '\*' || a[strlen(a) - 1] == '/') {

right = false;

}

else {

for (int i = 0; i < strlen(a); i++) {

if ((a[i] == '+' || a[i] == '-' || a[i] == '\*' || a[i] == '/') && (a[i + 1] == '+' || a[i + 1] == '\*' || a[i + 1] == '/')) {

right = false;

}

if ((a[i] == '/') && (a[i + 1] == '0')) {

right = false;

}

}

}

return right;

}

void ex1() {

cout << "1. Converting the entered expression: " << endl;

char a[100] = { 0 };

char out[100] = { 0 };

cout << "Expression: ";

gets\_s(a);

RPN(a, out);

cout << "RPN: " << out << endl << endl;

}

void ex2\_1() {

cout << endl << "2. 1)Checking for the correctness of a simple expression: " << endl;

char chek[100] = { 0 };

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Expression: ";

gets\_s(chek);

if (check(chek)) cout << "The expression is correct!" << endl;

else cout << "The expression is incorrect!" << endl;

}

void ex2\_2() {

cout << endl << " 2)Checking the RPN" << endl;

char chek[100] = { 0 };

//char cheked[100] = { 0 };

cout << "Expression: ";

gets\_s(chek);

//OPN(chek, cheked);

if (calc(chek)) cout << "The expression is correct!" << endl;

else cout << "The expression is incorrect!" << endl;

}

void ex3\_1() {

cout << endl << "3. 1)Evaluating a simple expression" << endl;

char a[100] = { 0 };

char out[100] = { 0 };

cout << "Expression: ";

gets\_s(a);

RPN(a, out);

cout << "Answer: " << calc(out) << endl << endl;

}

void ex3\_2() {

cout << endl << " 2)Evaluating a RPN" << endl;

char a[100] = { 0 };

cout << "Expression: ";

gets\_s(a);

cout << "Answer: " << calc(a) << endl << endl;

}

int main()

{

cout << "Expression input is implemented only from the keyboard" << endl << endl;

ex1();

ex2\_1();

ex2\_2();

ex3\_1();

ex3\_2();

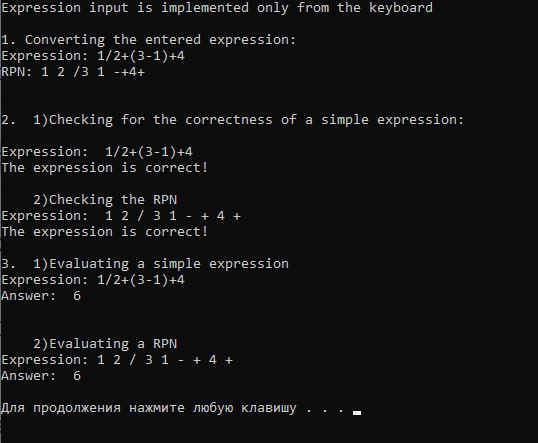
system("cls");

return 0;

}

# **ПРИЛОЖЕНИЕ B**

**РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

****