|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий |
|  |
| Кафедра Математического обеспечения и стандартизации информационных технологий |
|  | |
|  | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 7** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Применение стека и очереди при преобразовании и вычислении арифметических выражений»** | |
|  | |
| Выполнил студент группы ИКБО-09-22 | Гришин А. В. |
| Принял преподаватель | Филатов А.С. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

Москва 2023

1. Цель работы
   * 1. Получение знаний и практических навыков по работе с различными нотациями арифметических выражений.
2. Постановка задачи

Вариант 1.

1. Задание 1. Выполнить упражнения.
   1. Провести преобразование инфиксной записи выражения в префиксную нотацию, расписывая процесс по шагам.   
      S=a+(b-c\*k)/(d\*e-f)
   2. Представить инфиксную нотацию   
      выражений (идентификаторы односимвольные)   
      xyz\*abd/-+c-+  
      xyz\*d/+ab\*cok-/--
   3. Представить префиксную нотацию полученных в пункте 2 выражений
   4. Дана постфиксная запись арифметического выражения, представленного в строковом формате. Операнды однозначные числа. Провести вычисление значения выражения в постфиксной форме, расписывая процесс по шагам. 7 2 3 \* 5 8 2 / - + 1 - +
2. Задание 2. Выполнить программную реализацию следующих задач, используя структуру стек или очередь
   1. Разработать функцию преобразования инфиксной формы скобочного выражения в постфиксную форму.
   2. Реализовать класс стек, реализующий структуру и методы: втолкнуть элемент в стек, вытолкнуть элемент из стека, вернуть значение элемента в вершине стека, сделать стек пустым, определить пуст ли стек. Рассмотреть два варианта реализации: на массиве (или строке); на однонаправленном списке. Интерфейс программы должен обеспечивать непрерывную работу со структурой.
3. Решение
   1. Краткая теория по теме

Стек — это структура данных, которая работает по принципу FILO (first in — last out; первый пришел — последний ушел). В C++ уже есть готовый шаблон — stack.

В стеке элемент, который вошел самый первый — выйдет самым последним. Получается, если вы добавили три элемента в стек первым будет удален последний добавленный элемент.

Очередь — это структура данных (как было сказано выше), которая построена по принципу FILO (last in — last out: первым пришел — последним вышел). В C++ уже есть готовый STL контейнер — queue.

В очереди, если вы добавите элемент, который вошел первым, то он выйдет тоже самым первым. Получается, если вы добавите 4 элемента, то первый добавленный элемент выйдет первым.

Дек (deque — double ended queue, «двусторонняя очередь») – структура данных типа «список», функционирующая одновременно по двум принцам организации данных: FIFO и LIFO. Определить дек можно как очередь с двумя сторонами, так и стек, имеющий два конца. То есть данный подвид списка характерен двухсторонним доступом: выполнение поэлементной операции, определенной над деком, предполагает возможность выбора одной из его сторон в качестве активной.

Инфиксная нотация – оператор в выражениях расположен между (in between) двух операндов, с которыми он работает, например A+B.

Префиксная нотация – оператор расположен в выражениях перед операндами, с которыми он работает, например +AB, что в привычной для нас форме будет A+B

Постфиксная нотация – оператор расположен в выражениях после операндов, с которыми он работает, например AB+, что в привычной для нас форме будет A+B

* 1. Решение 1 задания

Преобразуем выражение инфиксной нотации в префиксную S = a+(b-c\*k)/(d\*e-f). Для этого расставим приоритет операций скобками и будем выносить операции за скобки перед операндами.

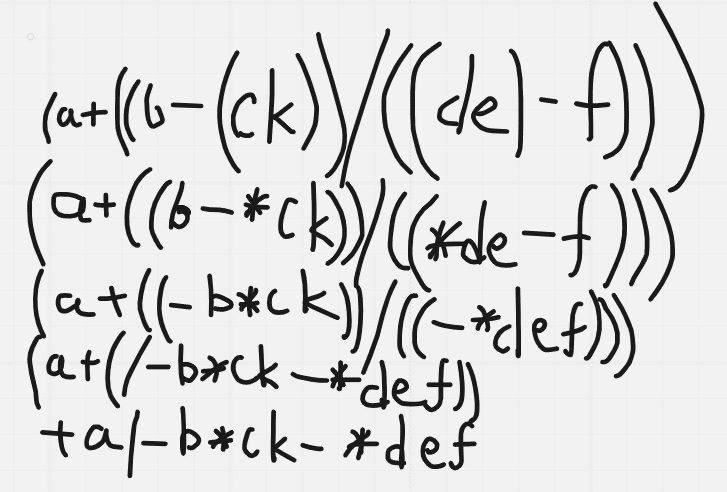


Рисунок 1. Преобразование инфиксной нотации в префиксную

Преобразуем выражения xyz\*abd/-+c-+ и xyz\*d/+ab\*cok-/-- из постфиксной нотации в инфиксную. Идем слева направо операнды добавляем в стек, а операторами действуем на два последних числа из стека.

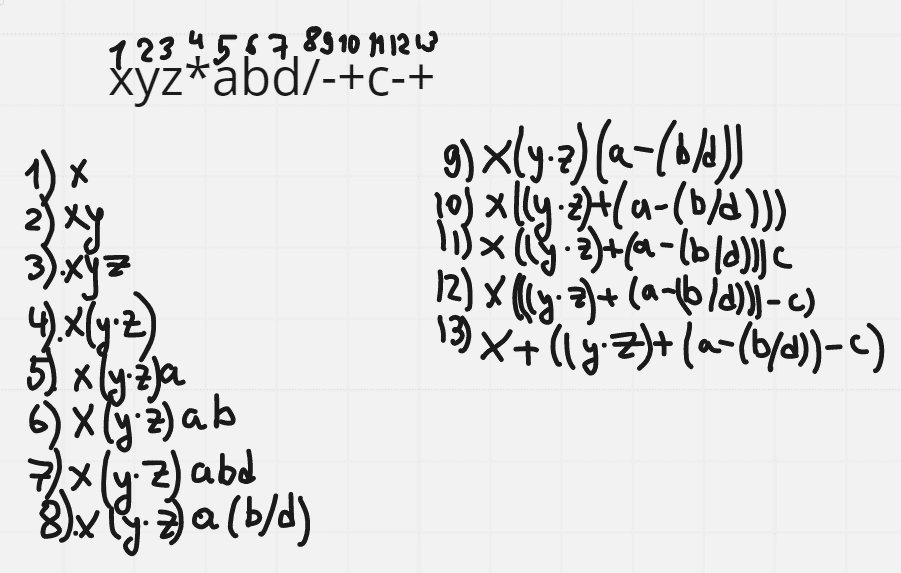


Рисунок 2. Преобразование постфиксной нотации в инфиксную

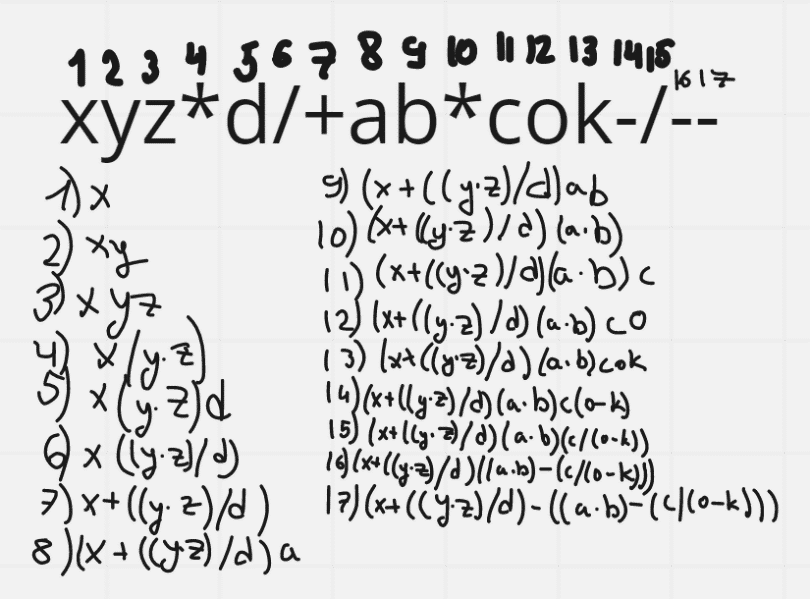


Рисунок 4. Преобразование постфиксной нотации в инфиксную

Преобразуем инфиксное выражения из прошлого номера в префиксные, предварительно расставив скобки.

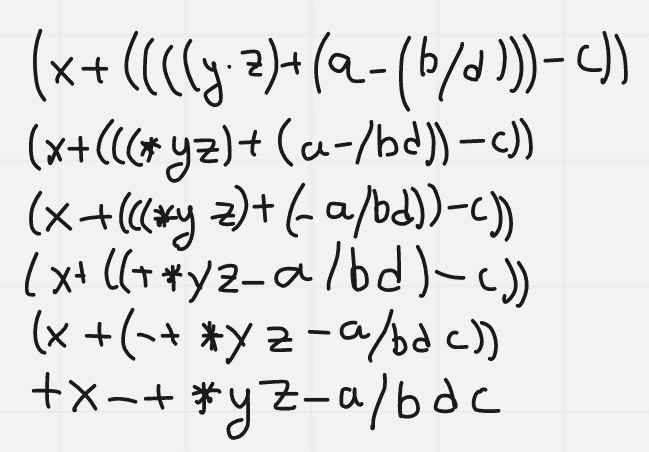


Рисунок 5. Преобразование инфиксной нотации в префиксную

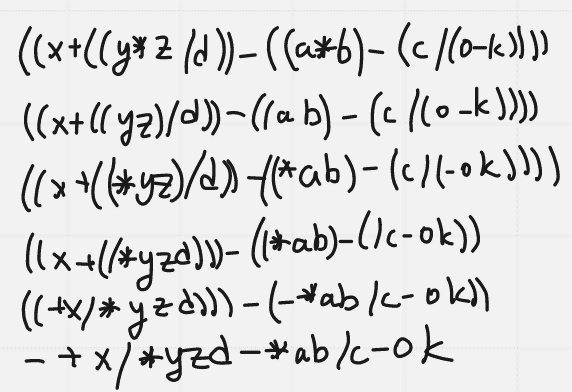


Рисунок 6. Преобразование инфиксной нотации в префиксную

Получим значения выражения (7 2 3 \* 5 8 2 / - + 1 - +), представленной в постфиксной форме.

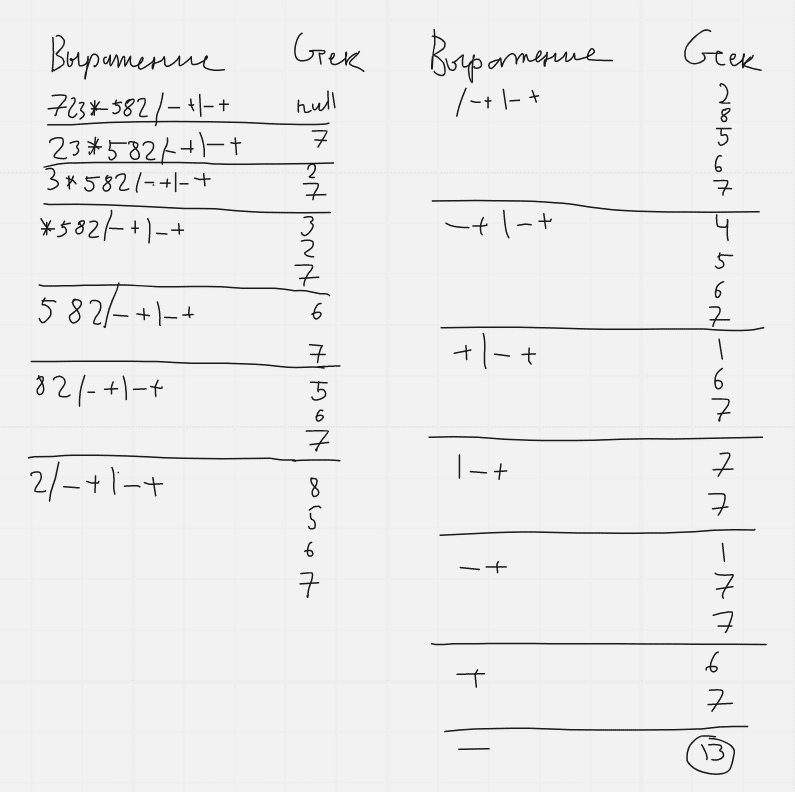


Рисунок 7. Вычисления значения выражения, представленного в постфиксной форме.

* 1. Решение 2 задания

Реализация задания была разбита на 3 файла. Основной – main.cpp (основной код программы), заголовочный – StackList.h (реализация класса стека на однонаправленном списке) и другой заголовочный – StackArray.h (реализация класса стека на статическом массиве).

Для реализации стека были написаны два шаблонных класса StackList и StackArray, реализующие стек на однонаправленном списке и статическом массиве соответственно.

|  |
| --- |
| template <typename T, int capacity>  class StackArray {  private:  T data[capacity];  int topIndex;  public:  StackArray() {  topIndex = -1;  }  ~StackArray() = default;  void push(T value) {  if (topIndex < capacity - 1) {  topIndex++;  data[topIndex] = value;  }  else {  cout << "Ошибка: переполнение стека\n";  }  }  void pop() {  if (topIndex >= 0) {  topIndex--;  }  }  T top() const {  if (topIndex >= 0) {  return data[topIndex];  }  return T();  }  bool isEmpty() const {  return topIndex == -1;  }  void clear() {  topIndex = -1;  }  void print() const {  if (isEmpty())  cout << "Стек пуст\n";  else {  cout << "Стек (сверху вниз):\n";  for (size\_t i = topIndex; i > 0; --i) {  cout << data[i] << "\n|\n";  }  cout << data[0] << "\n";  }  }  }; |

|  |
| --- |
| template <typename T>  class StackList {  private:  struct Node {  T data;  Node\* next;  };  Node\* topNode;  public:  StackList() {  topNode = nullptr;  }  ~StackList() {  while (!isEmpty()) {  pop();  }  }  void push(T value) {  Node\* newNode = new Node;  newNode->data = value;  newNode->next = topNode;  topNode = newNode;  }  void pop() {  if (!isEmpty()) {  Node\* tempNode = topNode;  topNode = topNode->next;  delete tempNode;  }  }  T top() const {  if (!isEmpty()) {  return topNode->data;  }  return T();  }  bool isEmpty() const {  return topNode == nullptr;  }  void clear() {  while (!isEmpty()) {  pop();  }  }  void print() const {  if (isEmpty()) {  cout << "Стек пуст\n";  }  else {  cout << "Стек (сверху вниз):\n";  Node\* node = topNode;  while (node != nullptr) {  cout << node->data;  if (node->next != nullptr)  cout << "\n|\n";  else  cout << "\n";  node = node->next;  }  }  }  }; |

Для удобства в шаблонном классе StackList была реализована структура узла списка Node.

|  |
| --- |
| struct Node {  T data;  Node\* next;  }; |

Для инициализации вершины стека в классах StackList и StackArray были реализованы конструкторы StackList и StackArray соответственно.

|  |
| --- |
| StackList() {  topNode = nullptr;  } |

|  |
| --- |
| StackArray() {  topIndex = -1;  } |

Для очистки памяти в классе StackList был реализован деструктор ~StackList.

|  |
| --- |
| ~StackList() {  while (!isEmpty()) {  pop();  }  } |

Для добавления элемента в стек была реализован метод push.

Метод для стека на списке.

|  |
| --- |
| void push(T value) {  Node\* newNode = new Node;  newNode->data = value;  newNode->next = topNode;  topNode = newNode;  } |

Метод для стека на статическом массиве.

|  |
| --- |
| void push(T value) {  if (topIndex < capacity - 1) {  topIndex++;  data[topIndex] = value;  }  else {  cout << "Ошибка: переполнение стека\n";  }  } |

Для извлечения верхнего элемента стека был реализован метод pop.

Метод для стека на списке.

|  |
| --- |
| void push(T value) {  Node\* newNode = new Node;  newNode->data = value;  newNode->next = topNode;  topNode = newNode;  } |

Метод для стека на статическом массиве.

|  |
| --- |
| void pop() {  if (topIndex >= 0) {  topIndex--;  }  } |

Для получения верхнего элемента стека был реализован метод pop.

Метод для стека на списке.

|  |
| --- |
| T top() const {  if (!isEmpty()) {  return topNode->data;  }  return T();  } |

Метод для стека на статическом массиве.

|  |
| --- |
| T top() const {  if (topIndex >= 0) {  return data[topIndex];  }  return T();  } |

Для поверки является ли список пустым был реализован метод isEmpty.

Метод для стека на списке.

|  |
| --- |
| bool isEmpty() const {  return topNode == nullptr;  } |

Метод для стека на статическом массиве.

|  |
| --- |
| bool isEmpty() const {  return topIndex == -1;  } |

Для очистки списка был реализован метод clear.

Метод для стека на списке.

|  |
| --- |
| void clear() {  while (!isEmpty()) {  pop();  }  } |

Метод для стека на статическом массиве.

|  |
| --- |
| void clear() {  topIndex = -1;  } |

Для вывода стека на экран был реализован метод print.

Метод для стека на списке.

|  |
| --- |
| void print() const {  if (isEmpty()) {  cout << " Стек пуст\n";  } else {  cout << "Стек (сверху вниз):\n";  Node \*node = topNode;  while (node != nullptr) {  cout << node->data;  if (node->next != nullptr)  cout << "\n|\n";  else  cout << "\n";  node = node->next;  }  }  } |

Метод для стека на статическом массиве.

|  |
| --- |
| void print() const {  if (isEmpty())  cout << "Стек пуст\n";  else {  cout << "Стек (сверху вниз):\n";  for (size\_t i = topIndex; i > 0; --i) {  cout << data[i] << "\n|\n";  }  cout << data[0] << "\n";  }  } |

Для проверки является ли символ оператор была реализована функция isOperator. Данная функция проверяет только четыре основных оператора.

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

Для проверки является ли символ операндом была реализована функция isOperand. Операндом в данном случае может быть любая буква латинского алфавита или число.

|  |
| --- |
| bool isOperand(char c) {  return (c >= '0' && c <= '9') || (c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z');  } |

Для определения приоритета операции была реализована функция precedence.

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

Перевод из инфиксной формы записи в постфиксную осуществляется функцией infixToPostfix, которая возвращает постфиксную форму записи выражения.

|  |
| --- |
| char\* infixToPostfix(const char\* infix) {  StackList<char> stack;  int length = 0;  for (size\_t i = 0; infix[i]; i++) {  char c = infix[i];  if (isOperand(c)) {  length++;  } else if (isOperator(c)) {  length += 2;  stack.push(c);  } else if (c == '(') {  stack.push(c);  } else if (c == ')') {  while (!stack.isEmpty() && stack.top() != '(') {  length++;  stack.pop();  }  if (!stack.isEmpty()) {  stack.pop();  }  }  }  while (!stack.isEmpty()) {  length++;  stack.pop();  }  char\* postfix = new char[length + 1];  int index = 0;  for (size\_t i = 0; infix[i]; i++) {  char c = infix[i];  if (isOperand(c)) {  postfix[index++] = c;  } else if (isOperator(c)) {  while (!stack.isEmpty() && stack.top() != '(' && precedence(c) <= precedence(stack.top())) {  postfix[index++] = stack.top();  stack.pop();  }  stack.push(c);  } else if (c == '(') {  stack.push(c);  } else if (c == ')') {  while (!stack.isEmpty() && stack.top() != '(') {  postfix[index++] = stack.top();  stack.pop();  }  if (!stack.isEmpty()) {  stack.pop();  }  }  }  while (!stack.isEmpty()) {  postfix[index++] = stack.top();  stack.pop();  }  postfix[index] = '\0';  return postfix;  } |

* 1. Описание работы программы

При запуске программы пользователю предлагается выбрать функционал, который он хочет использовать: 1 – Перевод из инфиксной формы записи в постфиксную. 2 – Работа со стеком.

При выборе первого варианта программа попросит ввести пользователя инфиксную форму записи выражения.

Если же выбрать второй вариант, то пользователя попросят выбрать вариант реализации стека для выполнения операций над ним. Далее в бесконечном цикле пользователь может выбирать любую операцию над стеком. Доступны следующие операции: втолкнуть элемент в стек, вытолкнуть элемент из стека, вернуть значение элемента в вершине стека, сделать стек пустым, определить пуст ли стек.

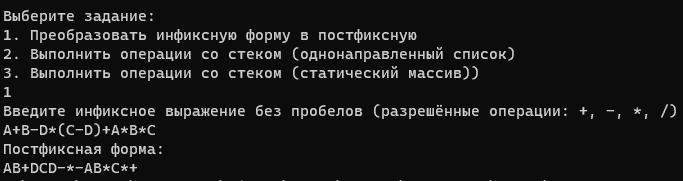


Рисунок 8. Интерфейс программы

1. Тестирование

Протестируем выполнение преобразования инфиксной записи выражения в постфиксную. Введем выражение A+B\*C-D

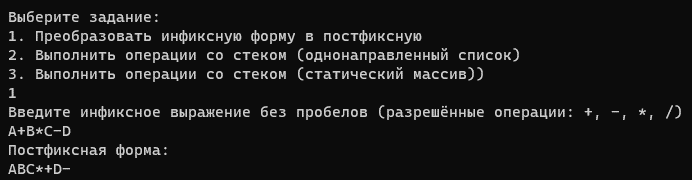


Рисунок 9. Тестирование преобразования инфиксной записи выражения в постфиксную

Как видно из рисунка 7, программа успешно определила приоритет операций и вывела верный ответ.

Протестируем выполнение преобразования инфиксной записи выражения в постфиксную, но теперь будет использовать скобки. Введем выражение (A+B)\*C-D/(J+H).

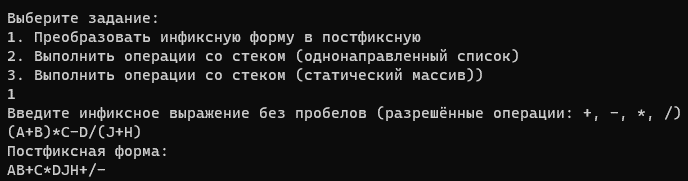


Рисунок 10. Тестирование преобразования инфиксной записи выражения в постфиксную

Как видно из рисунка 10, программа правильно определила приоритеты для выражений в скобках и операций, а также вывела верный ответ.

Протестируем выполнение операций над стеком используя реализацию стека на однонаправленном списке. Для этого для последовательно добавим элементы 1, 2, 3.

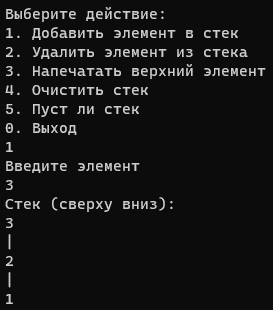


Рисунок 11. Тестирование операций над стеком, реализованном на однонаправленном списке

Из рисунка 11 видим, что элементы в стек добавились верно. Теперь попробуем проверить пуст ли стек.

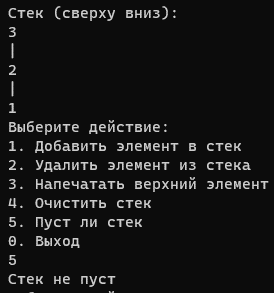


Рисунок 12. Тестирование операций над стеком, реализованном на однонаправленном списке

Действительно, стек не пуст, программа вывела верный ответ. Далее удалим верхним элемент стека и выведем на экран новый элемент на вершине стека.

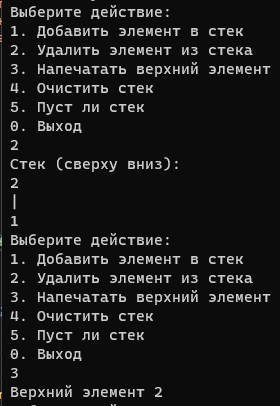


Рисунок 13. Тестирование операций над стеком, реализованном на однонаправленном списке

Исходя из рисунка 13 видим, что программа вывела верный ответ, верхний элемент списка успешно извлечен и теперь верхним должен стать не 3, а элемент 2, что и произошло. Теперь очистим стек.

Протестируем выполнение операций над стеком используя реализацию стека на статическом массиве. Для этого для последовательно добавим элементы 5, 4, 2, 8, -1.

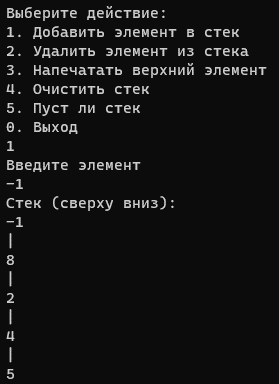


Рисунок 14. Тестирование операций над стеком, реализованном на однонаправленном списке

Из рисунка 14 видим, что элементы в стек добавились верно. Теперь попробуем проверить пуст ли стек.

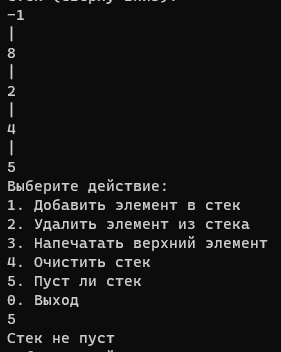


Рисунок 15. Тестирование операций над стеком, реализованном на однонаправленном списке

Действительно, стек не пуст, программа вывела верный ответ. Далее удалим верхним элемент стека и выведем на экран новый элемент на вершине стека.

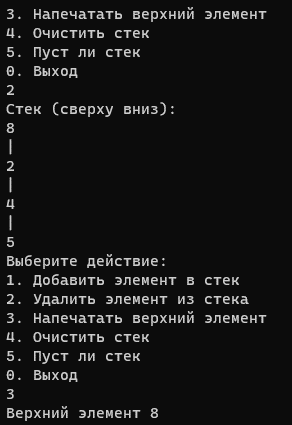


Рисунок 16. Тестирование операций над стеком, реализованном на однонаправленном списке

Исходя из рисунка 16 видим, что программа вывела верный ответ, верхний элемент списка успешно извлечен и теперь верхним должен стать не -1, а элемент 8, что и произошло. Теперь очистим стек.

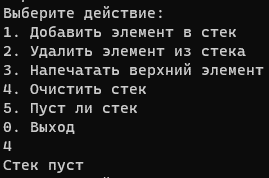


Рисунок 17. Тестирование операций над стеком, реализованном на однонаправленном списке

Как видно из рисунка 17, программа успешно очистила стек и теперь он пуст.

1. Вывод

В результате выполнения работы я получил знания и практические навыки по работе с различными нотациями арифметических выражений, а также работе со стеком и его реализацией.

1. Исходный код программы

Файл main.cpp

|  |
| --- |
| #include "StackList.h"  #include "StackArray.h"  #include <iostream>  using namespace std;  bool isOperator(char c) {  return c == '+' || c == '-' || c == '\*' || c == '/';  }  bool isOperand(char c) {  return (c >= '0' && c <= '9') || (c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z');  }  int precedence(char op) {  if (op == '+' || op == '-') {  return 1;  }  else if (op == '\*' || op == '/') {  return 2;  }  else {  return 0;  }  }  char\* infixToPostfix(const char\* infix) {  StackList<char> stack;  int length = 0;  for (size\_t i = 0; infix[i]; i++) {  char c = infix[i];  if (isOperand(c)) {  length++;  }  else if (isOperator(c)) {  length += 2;  stack.push(c);  }  else if (c == '(') {  stack.push(c);  }  else if (c == ')') {  while (!stack.isEmpty() && stack.top() != '(') {  length++;  stack.pop();  }  if (!stack.isEmpty()) {  stack.pop();  }  }  }  while (!stack.isEmpty()) {  length++;  stack.pop();  }  char\* postfix = new char[length + 1];  int index = 0;  for (size\_t i = 0; infix[i]; i++) {  char c = infix[i];  if (isOperand(c)) {  postfix[index++] = c;  }  else if (isOperator(c)) {  while (!stack.isEmpty() && stack.top() != '(' && precedence(c) <= precedence(stack.top())) {  postfix[index++] = stack.top();  stack.pop();  }  stack.push(c);  }  else if (c == '(') {  stack.push(c);  }  else if (c == ')') {  while (!stack.isEmpty() && stack.top() != '(') {  postfix[index++] = stack.top();  stack.pop();  }  if (!stack.isEmpty()) {  stack.pop();  }  }  }  while (!stack.isEmpty()) {  postfix[index++] = stack.top();  stack.pop();  }  postfix[index] = '\0';  return postfix;  }  int vibor() {  int n;  cout << "Практическое задание 7 по теме Применение стека и очереди при преобразовании и вычислении арифметических выражений. Гришин Андрей.\n";  cout << "Выберите задание: \n"  "1. Преобразовать инфиксную форму в постфиксную\n"  "2. Выполнить операции со стеком (однонаправленный список)\n"  "3. Выполнить операции со стеком (статический массив)\n";  cin >> n;  return n;  }  int main() {  setlocale(LC\_ALL, "RUS");  switch (vibor()) {  case 1: {  cout << "Введите инфиксное выражение без пробелов (разрешённые операции: +, -, \*, /)\n";  char s[100];  cin >> s;  cout << "Постфиксная форма:\n" << infixToPostfix(s);  break;  }  case 2: {  cout << "Стек хранит значения int\n";  StackList<int> stack;  int a = -1;  while (a != 0) {  cout << "Выберите действие: \n"  "1. Добавить элемент в стек\n"  "2. Удалить элемент из стека\n"  "3. Напечатать верхний элемент\n"  "4. Очистить стек\n"  "5. Пуст ли стек\n"  "0. Выход\n";  cin >> a;  switch (a) {  case 1: {  int value;  cout << "Введите элемент\n";  cin >> value;  stack.push(value);  stack.print();  break;  }  case 2: {  stack.pop();  stack.print();  break;  }  case 3: {  if (!stack.isEmpty())  cout << "Верхний элемент " << stack.top() << "\n";  else  stack.print();  break;  }  case 4: {  stack.clear();  stack.print();  break;  }  case 5: {  if (stack.isEmpty())  cout << "Стек пуст\n";  else  cout << "Стек не пуст\n";  break;  }  }  }  break;  }  case 3: {  cout << "Стек хранит значения int (макс. размер = 100)\n";  StackArray<int, 100> stack;  int a = -1;  while (a != 0) {  cout << "Выберите действие: \n"  "1. Добавить элемент в стек\n"  "2. Удалить элемент из стека\n"  "3. Напечатать верхний элемент\n"  "4. Очистить стек\n"  "5. Пуст ли стек\n"  "0. Выход\n";  cin >> a;  switch (a) {  case 1: {  int value;  cout << "Введите элемент\n";  cin >> value;  stack.push(value);  stack.print();  break;  }  case 2: {  stack.pop();  stack.print();  break;  }  case 3: {  if (!stack.isEmpty())  cout << "Верхний элемент " << stack.top() << "\n";  else  stack.print();  break;  }  case 4: {  stack.clear();  stack.print();  break;  }  case 5: {  if (stack.isEmpty())  cout << "Стек пуст\n";  else  cout << "Стек не пуст\n";  break;  }  }  }  break;  }  }  } |

Файл StackList.h

|  |
| --- |
| #pragma once  #include <iostream>  using namespace std;  template <typename T>  class StackList {  private:  struct Node {  T data;  Node\* next;  };  Node\* topNode;  public:  StackList() {  topNode = nullptr;  }  ~StackList() {  while (!isEmpty()) {  pop();  }  }  void push(T value) {  Node\* newNode = new Node;  newNode->data = value;  newNode->next = topNode;  topNode = newNode;  }  void pop() {  if (!isEmpty()) {  Node\* tempNode = topNode;  topNode = topNode->next;  delete tempNode;  }  }  T top() const {  if (!isEmpty()) {  return topNode->data;  }  return T();  }  bool isEmpty() const {  return topNode == nullptr;  }  void clear() {  while (!isEmpty()) {  pop();  }  }  void print() const {  if (isEmpty()) {  cout << "Стек пуст\n";  }  else {  cout << "Стек (сверху вниз):\n";  Node\* node = topNode;  while (node != nullptr) {  cout << node->data;  if (node->next != nullptr)  cout << "\n|\n";  else  cout << "\n";  node = node->next;  }  }  }  }; |

Файл StackArray.h

|  |
| --- |
| #pragma once  #include <iostream>  using namespace std;  template <typename T, int capacity>  class StackArray {  private:  T data[capacity];  int topIndex;  public:  StackArray() {  topIndex = -1;  }  ~StackArray() = default;  void push(T value) {  if (topIndex < capacity - 1) {  topIndex++;  data[topIndex] = value;  }  else {  cout << "Ошибка: переполнение стека\n";  }  }  void pop() {  if (topIndex >= 0) {  topIndex--;  }  }  T top() const {  if (topIndex >= 0) {  return data[topIndex];  }  return T();  }  bool isEmpty() const {  return topIndex == -1;  }  void clear() {  topIndex = -1;  }  void print() const {  if (isEmpty())  cout << "Стек пуст\n";  else {  cout << "Стек (сверху вниз):\n";  for (size\_t i = topIndex; i > 0; --i) {  cout << data[i] << "\n|\n";  }  cout << data[0] << "\n";  }  }  }; |