|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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-08-22 | Сенькевич Г.Д. |
| Принял преподаватель | Филатов А.С. |

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

Москва 2023

# **Цель работы**

Получение знаний и практических навыков по работе с различными нотациями арифметических выражений.

# **Постановка задачи**

Вариант №5. Условие задания:

|  |  |
| --- | --- |
| Упражнение 1 | Выполнить упражнения   1. Провести преобразование инфиксной записи выражения в префиксную нотацию, расписывая процесс по шагам   S=a+(b-c\*k)/(d\*e-f)   1. Представить инфиксную нотацию выражений (идентификаторы односимвольные)   xyz\*abd/-+c-+  xyz\*d/+ab\*cok-/--   1. Представить префиксную нотацию полученных в п2. выражений 2. 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 - + |
| Упражнение 2 | Выполнить программную реализацию следующих задач, используя структуру стек или очередь   1. Разработать функцию преобразования инфиксной формы скобочного арифметического выражения в префиксную форму 2. Реализовать класс стек на следующей структуре: хранить два стека в одном массиве, когда один располагается в начале массива и растет к концу массива, а второй располагается в конце и растет к началу. Реализуйте операцию Push(x,S) – втолкнуть элемент х в стек S, где S один или другой стек и соответствующую операцию Pop(S). Включите все необходимые проверки в эту процедуру. Интерфейс программы должен обеспечивать непрерывную работу со структурой. |

# **Решение**

Стэк.

Стек (stack) — это структура данных, которая представляет собой упорядоченный набор элементов, в котором доступны только две операции - добавление нового элемента и удаление последнего добавленного элемента. Также стек называют LIFO (Last-In-First-Out), что означает, что последний добавленный элемент будет первым удаленным.

Стек может быть реализован как статическая структура данных с фиксированным размером, так и как динамическая структура, которая может менять свой размер в зависимости от количества элементов. Одним из самых распространенных способов реализации стека является использование массива или связного списка.

Стек используется во многих алгоритмах и программах, так как позволяет эффективно решать ряд задач, связанных с обработкой данных. Одним из наиболее частых применений стека является обратная польская запись арифметических выражений. Также стек используется при реализации алгоритмов обхода деревьев, проверки правильности скобочных последовательностей и др.

Операции добавления и удаления элементов в стеке называются push и pop соответственно. Для реализации стека также используются операции top (получение значения верхнего элемента без удаления) и is\_empty (проверка, пуст ли стек).

Способы записи арифметических выражений.

Инфиксный формат записи выражения является наиболее распространенным и привычным для большинства людей. В инфиксной записи операторы записываются между операндами, например, 2 + 3. Однако, в инфиксной нотации может возникать неоднозначность, связанная с приоритетом операций, когда у нас есть несколько операторов с разным приоритетом. В таких случаях используются скобки, чтобы определить порядок операций.

Префиксный формат записи выражения (или польская нотация) является альтернативным способом записи арифметических выражений, где оператор записывается перед операндами, например, + 2 3. В этом случае не возникает неоднозначности, связанной с приоритетом операций, так как порядок операций задается явно. Однако, для непривычного восприятия префиксной нотации требуется некоторое время на привыкание.

Постфиксный формат записи выражения (или обратная польская нотация) предполагает запись оператора после операндов, например, 2 3 +. В этом случае порядок операций также задается явно, и не возникает неоднозначности, связанной с приоритетом операций. Постфиксная нотация также имеет преимущество в том, что ее легко интерпретировать компьютером при вычислении выражения.

Упражнение 1.

1. Для преобразования инфиксной записи выражения S в префиксную нотацию будем следовать следующим шагам:

Шаг 1: изменим порядок операндов и операторов в каждой группе скобок на обратный.

Было: S = a + ((b - c \* k) / (d \* e - f))

Стало: S = a + (/ (- b \* c k) (- d e f))

Шаг 2: перенесём операции деления и умножения из скобок в начало соответствующих выражений.

Было: S = a + (/ (- b c k) (- d e f))

Стало: S = a + / - \* b c k - \* d e f

Шаг 3: перенесём операцию сложения из начала выражения в самое начало всей записи.

Было: S = a + / - \* b c k - \* d e f

Стало: S = + a / - \* b c k - \* d e f

Таким образом, префиксная нотация выражения S будет -

"+ a / - \* b c k - \* d e f".

1. Первое выражение:

Постфиксная запись: xyz\*abd/-+c-+

Инфиксная запись: x + y \* z + a – b / d – c

Второе выражение:

Постфиксная запись: xyz\*d/+ab\*cok-/--

Инфиксная запись: x + y \* z / d - a \* b + c / (o - k)

1. Первое выражение:

Инфиксная запись: x + y \* z + a – b / d – c

Префиксная запись: - - + + x \* y z a / b d c

Второе выражение:

Инфиксная запись: x + y \* z / d - a \* b + c / (o - k)

Префиксная запись: + - + x / \* y z d \* a b / c - o k

1. Для вычисления значения выражения в постфиксной форме, будем следовать следующим шагам:

Шаг 1: создадим стек для хранения операндов.

Шаг 2: разобьём выражение на отдельные символы (операнды и операторы).

7 2 3 \* 5 8 2 / - + 1 - +

Шаг 3: пройдём по каждому символу в выражении.

Символ 7 является операндом, поэтому помещаем его в стек.

Символ 2 является операндом, поэтому помещаем его в стек.

Символ 3 является операндом, поэтому помещаем его в стек.

Символ \* является оператором. Извлекаем из стека два последних операнда (3 и 2), перемножаем их (3 \* 2 = 6) и помещаем результат (6) обратно в стек.

Символ 5 является операндом, поэтому помещаем его в стек.

Символ 8 является операндом, поэтому помещаем его в стек.

Символ 2 является операндом, поэтому помещаем его в стек.

Символ / является оператором. Извлекаем из стека два последних операнда (2 и 8), делим их (8 / 2 = 4) и помещаем результат (4) обратно в стек.

Символ - является оператором. Извлекаем из стека два последних операнда (5 и 4), вычитаем их (5 - 4 = 1) и помещаем результат (1) обратно в стек.

Символ + является оператором. Извлекаем из стека два последних операнда (6 и 1), складываем их (6 + 1 = 7) и помещаем результат (7) обратно в стек.

Символ 1 является операндом, поэтому помещаем его в стек.

Символ - является оператором. Извлекаем из стека два последних операнда (7 и 1), вычитаем их (7 - 1 = 6) и помещаем результат (6) обратно в стек.

Символ + является оператором. Извлекаем из стека два последних операнда (6 и 7), складываем их (6 + 7 = 13) и помещаем результат (13) обратно в стек.

Шаг 4: после обработки всех символов в выражении, результат вычисления будет находиться на вершине стека.

Таким образом, значение выражения – 13.

Для решения первого пункта второго упражнения была написана функция infix\_to\_prefix (а также для её работы реализован класс array\_stack (стэк на массиве)), выполняющая перевод арифметического выражения из инфиксной нотации в префиксную. Функция принимает на вход строку – выражение, которое нужно перевести, и возвращает строку – это выражение в префиксной нотации.

|  |
| --- |
| // Заголовочный файл класса array\_stack  #ifndef ARRAY\_STACK\_H  #define ARRAY\_STACK\_H  #include <iostream>  #define MAX\_ARRAY\_STACK\_SIZE 1000  class array\_stack  {  private:  int stack[MAX\_ARRAY\_STACK\_SIZE];  int top\_index;  public:  array\_stack();  bool is\_empty();  int top();  void push(int value);  int pop();  void clear();  };  #endif  // Функция infix\_to\_prefix  std::string infix\_to\_prefix(std::string expression)  {  array\_stack stack;  std::string prefix;  std::reverse(expression.begin(), expression.end());  for (char& ch : expression)  {  if (ch == '(' || ch == ')')  {  ch = (ch == '(' ? ')' : '(');  }  }  for (char ch : expression)  {  switch (ch)  {  case '(':  stack.push(ch);  break;  case ')':  while (stack.top() != '(')  {  prefix += stack.pop();  }  stack.pop();  break;  case '+':  case '-':  case '\*':  case '/':  case '^':  if (stack.is\_empty())  {  stack.push(ch);  }  else  {  if (stack.top() != '(')  {  while (precedence(ch) < precedence(stack.top()) ||  (precedence(ch) == precedence(stack.top()) &&  ch == '^'))  {  prefix += stack.pop();  if (stack.is\_empty())  {  break;  }  }  }  stack.push(ch);  }  break;  case ' ':  break;  default:  prefix += ch;  break;  }  }  while (!stack.is\_empty())  {  prefix += stack.pop();  }  std::reverse(prefix.begin(), prefix.end());  return prefix;  } |

Для решения второго пункта второго упражнения был написан класс double\_stack, реализующий все требуемые операции.

|  |
| --- |
| // Заголовочный файл класса double\_stack  #ifndef DOUBLE\_STACK\_H  #define DOUBLE\_STACK\_H  #include <iostream>  #define DOUBLE\_STACK\_SIZE 1000  enum side  {  left = 0,  right = 1  };  class double\_stack  {  private:  int storage[DOUBLE\_STACK\_SIZE];  int left\_stack\_last;  int right\_stack\_last;  public:  double\_stack();  bool is\_empty(side stack);  int top(side stack);  void push(side stack, int value);  int pop(side stack);  void clear(side stack);  };  #endif |

При запуске программы пользователь видит пользовательское меню, позволяющее выбрать между различными функциями, которые реализует программа.

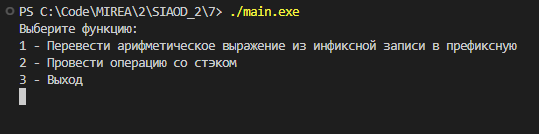


Рисунок 1. Интерфейс программы

# **Тестирование**

Протестируем выполнение программой первого пункта второго упражнения. Для этого выберем в меню первую функцию и введём выражение 7 \* (5 – 6), которое в префиксной нотации выглядит как \*7-56. На рисунке 2 видим, как программа вывела верный ответ.

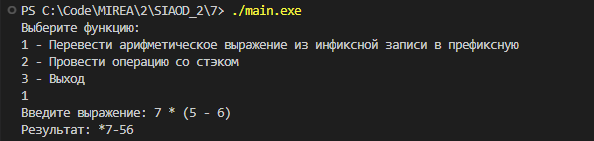


Рисунок 2. Тестирование программы

Протестируем выполнение программой второго пункта второго упражнения. Для этого выберем в меню вторую функцию, затем выберем левый стэк, положим в него 1, а после вызовем функцию, выводящую вершину стэка – там должно оказаться значение 1. На рисунке 3 видим, как программа вывела верный ответ.

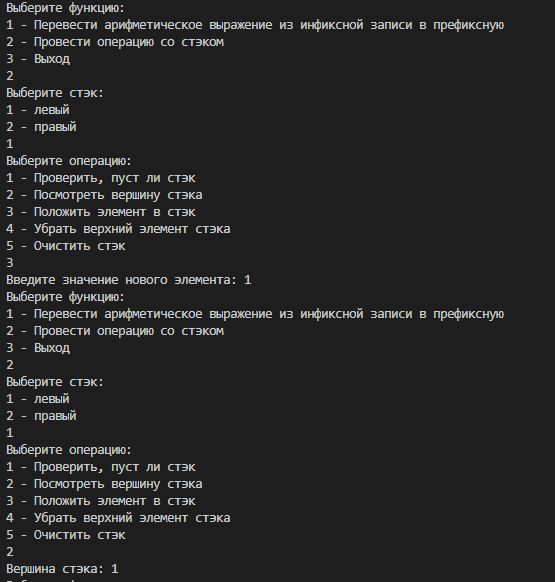


Рисунок 3. Тестирование программы

Из результатов тестирования видно, что программа работает корректно, решая все поставленные задачи.

# **Вывод**

В результате выполнения работы я:

1. Освоил алгоритмы работы с различными нотациями арифметических выражений;
2. Научился реализовывать стэк на языке программирования C++.

# **Исходный код программы**

|  |
| --- |
| // Файл main.cpp  #include <iostream>  #include <string>  #include <algorithm>  #include "double\_stack.h"  #include "array\_stack.h"  int precedence(char op)  {  switch (op)  {  case '+': case '-':  return 0;  case '\*': case '/':  return 1;  case '^':  return 2;  default:  return -1;  }  }  std::string infix\_to\_prefix(std::string expression)  {  array\_stack stack;  std::string prefix;  std::reverse(expression.begin(), expression.end());  for (char& ch : expression)  {  if (ch == '(' || ch == ')')  {  ch = (ch == '(' ? ')' : '(');  }  }  for (char ch : expression)  {  switch (ch)  {  case '(':  stack.push(ch);  break;  case ')':  while (stack.top() != '(')  {  prefix += stack.pop();  }  stack.pop();  break;  case '+':  case '-':  case '\*':  case '/':  case '^':  if (stack.is\_empty())  {  stack.push(ch);  }  else  {  if (stack.top() != '(')  {  while (precedence(ch) < precedence(stack.top()) ||  (precedence(ch) == precedence(stack.top()) &&  ch == '^'))  {  prefix += stack.pop();  if (stack.is\_empty())  {  break;  }  }  }  stack.push(ch);  }  break;  case ' ':  break;  default:  prefix += ch;  break;  }  }  while (!stack.is\_empty())  {  prefix += stack.pop();  }  std::reverse(prefix.begin(), prefix.end());  return prefix;  }  int main()  {  double\_stack stack;  while (true)  {  int choice = 0;  std::cout << "Выберите функцию:\n"  << "1 - Перевести арифметическое выражение из инфиксной"  << " записи в префиксную\n"  << "2 - Провести операцию со стэком\n"  << "3 - Выход\n";  std::cin >> choice;  if (choice == 1)  {  std::string expression;  std::cout << "Введите выражение: ";  std::cin.get();  char head = std::cin.get();  while (head != '\n')  {  expression += head;  head = std::cin.get();  }  std::cout << "Результат: " << infix\_to\_prefix(expression) << '\n';  }  else if (choice == 2)  {  int stack\_side\_choice = 0;  side stack\_side;  std::cout << "Выберите стэк:\n"  << "1 - левый\n"  << "2 - правый\n";  std::cin >> stack\_side\_choice;  if (stack\_side\_choice == 1)  {  stack\_side = left;  }  else if (stack\_side\_choice == 2)  {  stack\_side = right;  }  else  {  std::cout << "Такого варианта нет!\n";  continue;  }  int operation = 0;  std::cout << "Выберите операцию:\n"  << "1 - Проверить, пуст ли стэк\n"  << "2 - Посмотреть вершину стэка\n"  << "3 - Положить элемент в стэк\n"  << "4 - Убрать верхний элемент стэка\n"  << "5 - Очистить стэк\n";  std::cin >> operation;  if (operation == 1)  {  bool empty = stack.is\_empty(stack\_side);  std::cout << (empty ? "Стэк пуст" : "Стэк не пуст") << '\n';  }  else if (operation == 2)  {  if (!stack.is\_empty(stack\_side))  {  int value = stack.top(stack\_side);  std::cout << "Вершина стэка: " << value << '\n';  }  else  {  std::cout << "Стэк пуст\n";  }  }  else if (operation == 3)  {  int value = 0;  std::cout << "Введите значение нового элемента: ";  std::cin >> value;  stack.push(stack\_side, value);  }  else if (operation == 4)  {  if (!(stack.is\_empty(stack\_side)))  {  std::cout << "Удалённый элемент: "  << stack.pop(stack\_side) << '\n';  }  else  {  std::cout << "Стэк пуст\n";  }  }  else if (operation == 5)  {  stack.clear(stack\_side);  std::cout << "Стэк очищен\n";  }  else  {  std::cout << "Такого варианта нет!\n";  }  }  else if (choice == 3)  {  break;  }  else  {  std::cout << "Такого варианта нет!\n";  }  }  return 0;  }  // ФАЙЛ double\_stack.h  #ifndef DOUBLE\_STACK\_H  #define DOUBLE\_STACK\_H  #include <iostream>  #define DOUBLE\_STACK\_SIZE 1000  enum side  {  left = 0,  right = 1  };  class double\_stack  {  private:  int storage[DOUBLE\_STACK\_SIZE];  int left\_stack\_last;  int right\_stack\_last;  public:  double\_stack();  bool is\_empty(side stack);  int top(side stack);  void push(side stack, int value);  int pop(side stack);  void clear(side stack);  };  #endif  // ФАЙЛ array\_stack.h  #ifndef ARRAY\_STACK\_H  #define ARRAY\_STACK\_H  #include <iostream>  #define MAX\_ARRAY\_STACK\_SIZE 1000  class array\_stack  {  private:  int stack[MAX\_ARRAY\_STACK\_SIZE];  int top\_index;  public:  array\_stack();  bool is\_empty();  int top();  void push(int value);  int pop();  void clear();  };  #endif  // ФАЙЛ double\_stack.cpp  #include "double\_stack.h"  double\_stack::double\_stack()  {  left\_stack\_last = -1;  right\_stack\_last = DOUBLE\_STACK\_SIZE;  }  bool double\_stack::is\_empty(side stack)  {  if (stack == left)  {  return left\_stack\_last < 0;  }  else if (stack == right)  {  return right\_stack\_last >= DOUBLE\_STACK\_SIZE;  }  else  {  std::cerr << "Invalid stack parameter\n";  throw;  }  }  int double\_stack::top(side stack)  {  if (stack == left)  {  if (!(is\_empty(stack)))  {  return storage[left\_stack\_last];  }  else  {  std::cerr << "Stack is empty\n";  throw;  }  }  else if (stack == right)  {  if (!(is\_empty(stack)))  {  return storage[right\_stack\_last];  }  else  {  std::cerr << "Stack is empty\n";  throw;  }  }  else  {  std::cerr << "Invalid stack parameter\n";  throw;  }  }  void double\_stack::push(side stack, int value)  {  if (stack == left)  {  if (left\_stack\_last + 1 < right\_stack\_last)  {  left\_stack\_last++;  storage[left\_stack\_last] = value;  }  else  {  std::cerr << "Stack is full\n";  throw;  }  }  else if (stack == right)  {  if (right\_stack\_last - 1 > left\_stack\_last)  {  right\_stack\_last--;  storage[right\_stack\_last] = value;  }  else  {  std::cerr << "Stack is full\n";  throw;  }  }  else  {  std::cerr << "Invalid stack parameter\n";  throw;  }  }  int double\_stack::pop(side stack)  {  if (stack == left)  {  if (!(is\_empty(stack)))  {  left\_stack\_last--;  return storage[left\_stack\_last + 1];  }  else  {  std::cerr << "Stack is empty\n";  throw;  }  }  else if (stack == right)  {  if (!(is\_empty(stack)))  {  right\_stack\_last++;  return storage[right\_stack\_last - 1];  }  else  {  std::cerr << "Stack is empty\n";  throw;  }  }  else  {  std::cerr << "Invalid stack parameter\n";  throw;  }  }  void double\_stack::clear(side stack)  {  if (stack == left)  {  left\_stack\_last = -1;  }  else if (stack == right)  {  right\_stack\_last = DOUBLE\_STACK\_SIZE;  }  else  {  std::cerr << "Invalid stack parameter\n";  throw;  }  }  // ФАЙЛ array\_stack.cpp  #include "array\_stack.h"  array\_stack::array\_stack()  {  top\_index = -1;  }  bool array\_stack::is\_empty()  {  return top\_index < 0;  }  int array\_stack::top()  {  if (!is\_empty())  {  return stack[top\_index];  }  else  {  std::cerr << "top() : stack is empty\n";  throw "top() : stack is empty";  }  }  void array\_stack::push(int value)  {  if (top\_index + 1 < MAX\_ARRAY\_STACK\_SIZE)  {  top\_index++;  stack[top\_index] = value;  }  else  {  std::cerr << "push() : stack overflow\n";  throw "push() : stack overflow";  }  }  int array\_stack::pop()  {  if (!is\_empty())  {  top\_index--;  return stack[top\_index + 1];  }  else  {  std::cerr << "pop() : stack is empty\n";  throw "pop() : stack is empty";  }  }  void array\_stack::clear()  {  top\_index = -1;  } |