**Отчёт по лабораторной работе №1**

**Линейные списки**

Задание:

Выполнить задания, используя для представления очередей и стеков:

а) массивы;

б) динамические списки.

Требования к программам:

1. Количество элементов исходных линейных списков заранее не определено

и задается случайным образом. При дальнейшей обработке считается, что

количество элементов списка не известно, т.е. обработка производится, пока

не достигнут конец списка.

2. Программа должна сформировать исходные линейные списки, вывести их содержимое на экран (при этом данные из списков не должны быть потеряны), произвести

обработку и вывести содержимое итогового списка на экран.

14. Даны две очереди целых чисел. Сформировать стек из элементов первой

очереди, кратных минимуму второй.

Код программы:

|  |
| --- |
| #include <iostream>  #include <random>  class Queue {  private:  struct Node {  int data;  Node\* next;  };  Node\* head, \* tail;  public:  Queue() {  head = NULL;  tail = NULL;  }  bool is\_empty() const {  return head == NULL;  }  void print() const {  if (!is\_empty()) {  Node\* tmp = head;  while (tmp->next) {  std::cout << tmp->data << " ";  tmp = tmp->next;  }  std::cout << tmp->data << std::endl;  }  else {  std::cout << "Empty" << std::endl;  }  }  void add(int value) {  if (is\_empty()) {  head = new Node;  head->data = value;  head->next = NULL;  tail = head;  }  else {  tail->next = new Node;  tail = tail->next;  tail->data = value;  tail->next = NULL;  }  }  int remove() {  if (is\_empty()) {  std::cout << "Queue is empty";  return NULL;  }  int d = head->data;  Node\* tmp = head;  head = head->next;  delete(tmp);  return d;  }  void clear() {  Node\* tmp;  while (!is\_empty()) {  tmp = head;  head = head->next;  delete(tmp);  }  }  };  class ArrayQueue {  private:  int head, tail, size;  int\* data;  int next(int n) const {  return (n + 1) % size;  }  public:  ArrayQueue(int capacity) {  size = capacity + 1;  data = new int[size];  clear();  }  bool is\_empty() const {  return next(tail) == head;  }  void print() const {  if (!is\_empty()) {  for (int i = 0; i < next(tail); i++)  std::cout << data[i] << " ";  std::cout << std::endl;  }  else {  std::cout << "Empty" << std::endl;  }  }  void add(int value) {  if (next(next(tail)) == head) {  std::cout << "Queue overflow";  }  else {  tail = next(tail);  data[tail] = value;  }  }  int remove() {  if (is\_empty()) {  std::cout << "Queue is empty";  return NULL;  }  int d = data[head];  head = next(head);  return d;  }  void clear() {  head = 0;  tail = size - 1;  }  };  class Stack {  private:  struct Node {  int data;  Node\* next;  };  Node\* top;  public:  Stack() {  top = NULL;  }  bool is\_empty() const {  return top == NULL;  }  void print() const {  if (!is\_empty()) {  Node\* tmp = top;  while (tmp->next) {  std::cout << tmp->data << " ";  tmp = tmp->next;  }  std::cout << tmp->data << std::endl;  }  else {  std::cout << "Empty" << std::endl;  }  }  void push(int value) {  Node\* tmp = new Node;  tmp->next = top;  top = tmp;  top->data = value;  }  int pop() {  if (!is\_empty()) {  Node\* tmp = top;  int d = top->data;  top = top->next;  delete(tmp);  return d;  }  return NULL;  }  void clear() {  Node\* tmp;  while (!is\_empty()) {  tmp = top;  top = top->next;  delete(tmp);  }  }  };  class ArrayStack {  private:  int top;  int\* data;  int used;  int capacity;  public:  ArrayStack(int size) {  data = new int[size];  capacity = size;  top = -1;  }  bool is\_empty() const {  return top == -1;  }  bool is\_full() const {  return top >= capacity;  }  void print() const {  if (!is\_empty()) {  for (int i = top - 1; i >= 0; i--)  std::cout << data[i] << " ";  std::cout << std::endl;  }  else {  std::cout << "Empty" << std::endl;  }  }  void push(int value) {  if (is\_full()) {  std::cout << "Stack overflow!";  }  else {  data[++top] = value;  }  }  int pop() {  if (!is\_empty()) {  return data[top--];  }  return NULL;  }  void clear() {  top = -1;  }  };  // =========================================================================  // Динамическими списками  void task(Queue queue1, Queue queue2, Stack stack) {  // Заполнение  for (int i = 0; i < rand() % 1000; i++)  queue1.add(rand() % 100);  for (int i = 0; i < rand() % 1000; i++)  queue2.add(rand() % 100);  // Вывод очередей на экран  std::cout << " - Queue 1: ";  queue1.print();  std::cout << " - Queue 2: ";  queue2.print();  // Поиск минимума  int min = queue2.remove();  while (!queue2.is\_empty()) {  int elem = queue2.remove();  if (elem < min)  min = elem;  }  std::cout << " - Queue 2 min: " << min << std::endl;  // Формирование стека по условию  while (!queue1.is\_empty()) {  int elem = queue1.remove();  if (min != 0) {  if (elem % min == 0)  stack.push(elem);  }  }  std::cout << " - Stack: ";  stack.print();  }  // Массивами  void task(ArrayQueue queue1, ArrayQueue queue2, ArrayStack stack) {  // Заполнение  for (int i = 0; i < rand() % 1000; i++)  queue1.add(rand() % 100);  for (int i = 0; i < rand() % 1000; i++)  queue2.add(rand() % 100);  // Вывод очередей на экран  std::cout << " - Queue 1: ";  queue1.print();  std::cout << " - Queue 2: ";  queue2.print();  // Поиск минимума  int min = queue2.remove();  while (!queue2.is\_empty()) {  int elem = queue2.remove();  if (elem < min)  min = elem;  }  std::cout << " - Queue 2 min: " << min << std::endl;  // Формирование стека по условию  while (!queue1.is\_empty()) {  int elem = queue1.remove();  if (min != 0) {  if (elem % min == 0)  stack.push(elem);  }  }  std::cout << " - Stack: ";  stack.print();  }  int main() {  srand(time(0));  // Динамическими списками  std::cout << "\* Dynamic lists:" << std::endl;  Queue q1;  Queue q2;  Stack s;  task(q1, q2, s);  std::cout << std::endl;  // Массивами  std::cout << "\* Arrays:" << std::endl;  ArrayQueue aq1 = ArrayQueue(1000);  ArrayQueue aq2 = ArrayQueue(1000);  ArrayStack as = ArrayStack(1000);  task(aq1, aq2, as);  return 0;  } |

Алгоритм работы:

Программа состоит из двух частей – первая часть использует реализацию очереди и стека при помощи динамического списка, а вторая – при помощи одномерного массива.

В первой части программы создаётся новые объекты – две очереди и стек. Очереди заполняются случайным количеством случайных целых чисел, а затем выводятся на экран при помощи метода print(). Затем во второй очереди ищется минимальный элемент: пока она не пуста, из неё извлекается элемент при помощи метода remove() и сравнивается с предыдущим. Полученный минимум выводится на экран. Затем заполняется созданный ранее стек: пока первая очередь не окажется пустой из неё извлекается элемент и проверяется на кратность полученному минимуму. Если условие выполняется – элемент вносится в стек при помощи метода push(). После обработки всех элементов очереди стек выводится на экран при помощи метода print().

Во второй части программы также создаётся две очереди и стек, но реализованные при помощи одномерного массива размером 1000 элементов. При этом алгоритм обработки созданных объектов остаётся идентичным.

Метод Queue.print(): Первоначально функция проверяет не пуста ли очередь – если пуста, то печатается сообщение «Empty» и функция прекращает работу. Если же очередь не пуста, то создаётся временный объект Node, в который, согласно определению очереди (FIFO), вносится «голова» очереди. Происходит вход в цикл, выполняться который будет до тех пор, пока у элемента очереди существует указатель на следующий элемент. В цикле же на экран печатается значение, хранящееся во временном объекте Node, а затем ему присваивается следующий элемент очереди. Выполнение функции завершается выводом на экран «хвоста» очереди.

Метод ArrayQueue.print(): Первоначально функция проверяет не пуста ли очередь – если пуста, то печатается сообщение «Empty» и функция прекращает работу. Если же очередь не пуста, то производится классический вывод элементов массива на экран при помощи цикла. Цикл работает до тех пора пока не будет достигнут «хвост» очереди.

Метод Stack.print(): Первоначально функция проверяет не пуста ли очередь – если пуста, то печатается сообщение «Empty» и функция прекращает работу. Если же очередь не пуста, то создаётся временный объект Node, в который, согласно определению очереди (LIFO), вносится верхушка очереди. Происходит вход в цикл, выполняться который будет до тех пор, пока у элемента стека существует указатель на следующий элемент. В цикле же на экран печатается значение, хранящееся во временном объекте Node, а затем ему присваивается следующий элемент очереди. Выполнение функции завершается выводом на экран последнего элемента стека.

Метод ArrayStack.print(): Первоначально функция проверяет не пуста ли очередь – если пуста, то печатается сообщение «Empty» и функция прекращает работу. Если же очередь не пуста, то производится классический вывод элементов массива на экран при помощи цикла. При этом выполнение цикла начинается с верхушки стека и происходит до достижения последнего элемента стека.

Результат работы:



