Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования «Томский государственный университет систем
управления и радиоэлектроники»

Факультет систем управления (ФСУ)

Кафедра автоматизированных систем управления (АСУ)

Стеки и очереди
отчет по лабораторной работе №2 по дисциплине
«Структуры и алгоритмы обработки данных в ЭВМ»

Обучающийся гр. 431-3
_____ Сергиевский Д.В.
«_11_» _октября____ 2022 г.

Проверил: доцент каф. АСУ, д.т.н.
_____Горитов А. Н.
«_11_» _октября____ 2022 г.

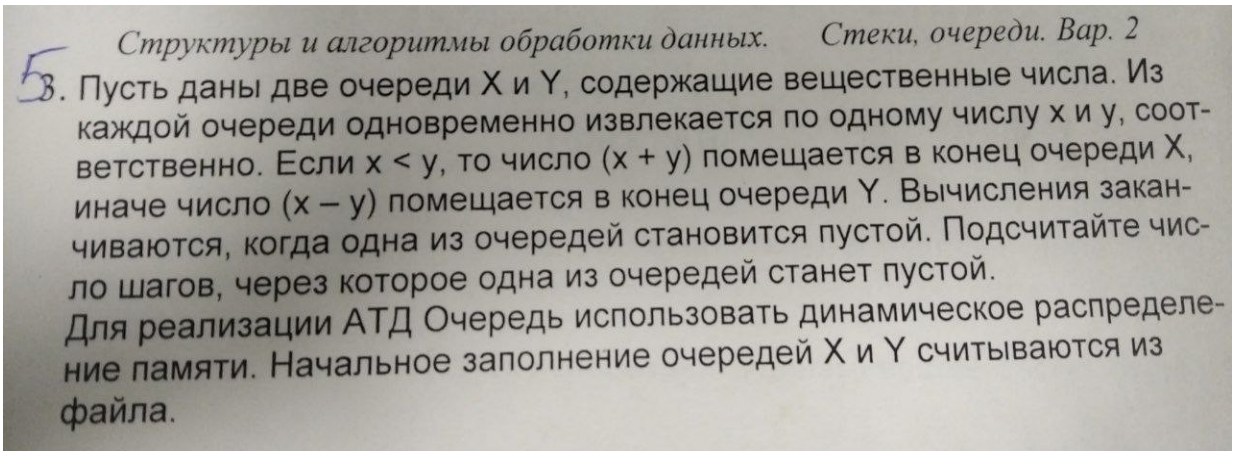
Томск 2022

Содержание

Введение

В рамках данной лабораторной работы необходимо решить небольшую задачу с применением АТД Очередь для закрепления теоретического материала.

Задание на лабораторную работу представлено на Рисунке 1.



Структуры и алгоритмы обработки данных. Стеки, очереди. Вар. 2

5

3. Пусть даны две очереди X и Y, содержащие вещественные числа. Из каждой очереди одновременно извлекается по одному числу x и y, соответственно. Если x < y, то число (x + y) помещается в конец очереди X, иначе число (x – y) помещается в конец очереди Y. Вычисления заканчиваются, когда одна из очередей становится пустой. Подсчитайте число шагов, через которое одна из очередей станет пустой.
Для реализации АТД Очередь использовать динамическое распределение памяти. Начальное заполнение очередей X и Y считываются из файла.

Рисунок 1 - задание

1 Ход работы

1.1 Алгоритм решения

Алгоритм решения задачи в полной мере описан в условии задачи.

Поскольку формат входных данных для инициализации очереди не определен в тексте задания, в качестве оного был принят произвольный: [n x1 x2 … xn], где n – количество элементов очереди, x1..xn. - элементы очереди.

В случае неверных входных данных программа завершается с выводом пояснительного сообщения.

1.2 Реализация решения

Для решения данной задачи потребовалось написать небольшую программу, реализующую вышеописанный алгоритм, а также АТД Очередь на основе динамического распределения памяти в качестве вспомогательного компонента.

Листинг программы приведен в Приложении А.

1.3 Пример решения

В качестве примера рассмотрим поведение программы на входных данных, представленных на Рисунке 1.1.

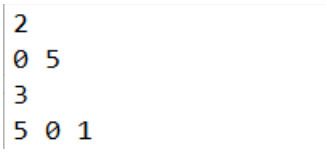


Рисунок 1.1 - пример входных данных

После завершения инициализации очереди будут заполнены следующими значениями: X[0 5], Y[5 0 1]. Далее наполнение очередей будет итерационно изменяться в соответствии с изложенной в задании логикой, во время чего будет подсчитываться количество пройденных итераций.

- На первом шаге из очередей будут извлечены значения x=0 и y=5, затем, поскольку 0<5, в конец очереди X будет записано значение 0+5=5, после чего очереди будут содержать следующие значения: X[5 5], Y[0 1]
- На втором шаге: (x=5 ≥ y=0) => Y ← (5-0=5) : X[5], Y[1 5]
- На третьем шаге: (x=5 ≥ y=1) => Y ← (5-1=4) : X[], Y[5]

  По завершении третьего шага очередь X окажется пустой, в связи с чем алгоритм завершит работу с выводом числа пройденных шагов.

  Пример вывода программы представлен на Рисунке 1.2.

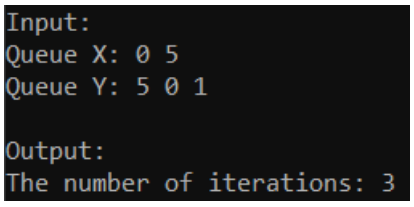


Рисунок 1.2 - пример вывода

Вывод

В результате данной лабораторной работы были подкреплены теоретические знания по теме «АТД Очередь» реализацией соответствующей структуры и её применением для решения небольшой задачи.

Приложение А

Файл s3_sad_lab2.cpp. Точка входа в программу и решение задачи.

```
// s3_sad_lab2.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.
//

#include <iostream>
#include <fstream>
#include <string>
using namespace std;

using val_type = double;

class QueueNode
{
private:
    val_type value;
    QueueNode* next;

    friend class Queue;

public:
    QueueNode(val_type value, QueueNode* next = nullptr) : value(value), next(next)
    {}
};

class Queue
{
private:
    QueueNode* head;
    QueueNode* tail;
```

```
public:

   Queue()
   {
      head = nullptr;
      tail = nullptr;
   }

   ~Queue()
   {
      while (!is_empty())
         pop();
   }

   void push(val_type val)
   {
      if (head == nullptr)
      {
         head = tail = new QueueNode(val);
         if (head == nullptr)
            throw "new returned nullptr";
         return;
      }

      tail = tail->next = new QueueNode(val);
      if (tail == nullptr)
         throw "new returned nullptr";
   }

   val_type pop()
   {
      if (is_empty())
```

```
        {
            throw "pop() when queue is empty";
        }

        val_type res = head->value;

        delete exchange(head, head->next);
        if (head == nullptr)
            tail = nullptr;

        return res;
    }

    val_type front() const
    {
        if (is_empty())
        {
            throw "front() when queue is empty";
        }

        return head->value;
    }

    bool is_empty() const
    {
        return head == nullptr;
    }
};

void do_task(string filename)
{
    fstream input(filename, fstream::in);
    if (!input.is_open())
```

```
{
   throw filename + " didn`t open";
}

// init

Queue X = Queue();
Queue Y = Queue();

size_t length;
double buffer;

cout << "Input: " << endl;

cout << "Queue X: ";
input >> length;
for (size_t i = 0; i < length; ++i)
{
   input >> buffer;
   X.push(buffer);

   cout << buffer << " ";
}
cout << endl;

cout << "Queue Y: ";
input >> length;
for (size_t i = 0; i < length; ++i)
{
   input >> buffer;
   Y.push(buffer);

   cout << buffer << " ";
```

```
    }
    cout << endl;

    cout << endl;

    // do

    cout << "Output: " << endl;

    size_t count = 0;

    while (!X.is_empty() && !Y.is_empty())
    {
        if (X.front() < Y.front())
            X.push(X.pop() + Y.pop());
        else
            Y.push(X.pop() - Y.pop());

        ++count;
    }

    cout << "The number of iterations before any queue is empty: " << count << endl;
}

int main()
{
    try
    {
        do_task("input.txt");
    }
    catch (const char* message)
    {
        cout << "error: " << message;
```

```
        getchar();
    }
}
```