МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

Тема: Линейные структуры данных: стек, очередь и дек

Студент гр. 7383 _____ Тян Е.

Преподаватель _____ Размочаева Н. В.

Санкт-Петербург
2018

СОДЕРЖАНИЕ

1. ЦЕЛЬ РАБОТЫ 3

2. РЕАЛИЗАЦИЯ ЗАДАЧИ 4

3. ТЕСТИРОВАНИЕ 7

4. ВЫВОД 8

ПРИЛОЖЕНИЕ А. ТЕСТОВЫЕ СЛУЧАИ 9

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД ПРОГРАММЫ 14

1. ЦЕЛЬ РАБОТЫ

Цель работы: ознакомиться с часто используемыми на практике линейными структурами данных, обеспечивающими доступ к элементам последовательности только через её начало и конец, и способами реализации этих структур, освоить на практике использование очереди для решения задач.

Формулировка задачи: рассматриваются следующие типы данных:

type *имя* = (*Анна*, …, *Яков*);

дети = **array**[*имя*, *имя*] **of** Boolean;

потомки = **file of** *имя*.

Задан массив *Д* типа *дети* (*Д*[*x*, *y*] = *true* если человек по имени *y* является ребенком человека по имени *x*). Для введенного пользователем имени *И* записать в файл *П* типа потомки имена всех потомков человека с именем *И* в следующем порядке: сначала имена всех его детей, затем всех его внуков, затем всех его правнуков и т.д.

Входные данные: Последовательность пар [*имя*, *имя*] = *true*.

2. РЕАЛИЗАЦИЯ ЗАДАЧИ

В данной работе используются главная функция (int main()) и дополнительные функции (char** list_of_names(FILE* ptr, int num, bool** children), int isempty(queue *q), queue* init(), void push(queue* q, int x), void pop(queue* q, char** name, FILE* exit)).

Параметры передаваемые в функцию char** list_of_names(FILE* ptr, int num, bool** children):

- ptr – указатель, на место в файле, где остановилось считывание;
- num – значение, показывающее, сколько человек будет в списке;
- children – массив типа boolean, хранящий пары [родитель; ребенок].

Параметры передаваемые в функцию int isempty(queue *q):

- q – очередь.

Параметры передаваемые в функцию void push(queue* q, int x):

- q – очередь;
- x – индекс родителя, которого нужно записать в очередь.

Параметры передаваемые в функцию void pop(queue* q, char** name, FILE* exit):

- q – очередь;
- name – массив, хранящий список имен;
- exit – указатель, на место в файле, где остановилась запись.

В функции main() выводится строка: "Choose file" и пользователь вводит название файла, выбранное из уже существующих. Если название файла было указано правильно и файл не пуст, создаются массивы: один типа boolean, хранящий набор 0, и второй, пустой массив, который будет хранить список имен. Вызывается функция list_of_names(FILE* ptr, int num, bool** children), которая считывает из файла пары и записывает их в список имен и массив типа boolean одновременно. В функции list_of_names(FILE* ptr, int num, bool** children) считывается первая строка из файла, выбранного пользователем, и первое имя записывается в фиктивную строку и, если этого имени нет в списке имен, записывается в список имен. Далее фиктивная

строка опустошается и в неё записывается второе имя, если этого имени не было в списке, то оно также записывается в список имен, после считывается число, которое должно быть 1, если все верно, то в массив типа boolean на пересечении строки первого имени и столбца второго имени ставиться значение true. Таким образом функция продолжает свою работу пока не дойдет до конца файла. По возвращении в функцию main() на консоль выводится список имен и массив типа boolean. Файл, откуда происходило считывание, закрывается. Создается очередь с помощью функции queue* init(): выделяется память под структуру очереди и в значение индекса начала очереди записывается 1, а в конец – 0. Далее пользователя просят ввести имя человека, чьих потомков нужно вывести в новый файл. Считывается введенное имя, если такого не существует в списке имен, то выводится ошибка и программа завершает свою работу. Если такое имя есть, то его индекс из списка имен записывается в очередь при вызове функции push(queue* q, int x): в ячейку массива – очереди записывается индекс, значение индекса конца очереди увеличивается на единицу. Далее имя первого человека в очереди проверяется на наличие детей, если у него такие есть, то их индексы записывают в очередь, и происходит очищение очереди от этого имени при помощи функции pop(queue* q, char** name, FILE* exit): в файл, куда указывает exit, записывается имя первого человека в очереди, потом все индексы находящиеся в очереди "сдвигаются" на один вперед и указатель на конец очереди уменьшается на единицу. Все продолжается пока очередь не станет пустой.

Для более понятного описания работы программы рассмотрим случай работы программы. Пусть пользователь выбрал файл names4.txt. После ввода названия файла программа начинает считывание имен из файл и записывает их в список имен и заполняет массив типа boolean, который представлен в табл. 1.

Далее пользователь вводит имя. Допустим пользователь ввел имя Hamilton. Программа считывает имя и записывает в очередь индексы детей

Таблица 1 — Массив содержащий связи пар [родитель, ребенок]

	Amelia	Bethany	Charlotte	David	Evian	Hamilton	Mia	George	Sophia
Amelia	0	1	1	0	0	0	0	0	0
Bethany	0	0	0	0	0	0	0	0	0
Charlotte	0	0	0	1	1	0	0	0	0
David	0	0	0	0	0	0	0	0	0
Evian	0	0	0	0	0	0	0	0	0
Hamilton	0	0	0	0	0	0	1	1	0
Mia	0	0	0	0	0	0	0	0	0
George	0	0	0	0	1	0	0	0	1
Sophia	0	0	0	0	0	0	0	0	0

родителя, данный шаг приведен на рис. 1. Далее происходит «выход» первого

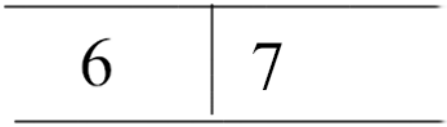

Рисунок 1 — очередь, в которой записаны только дети

ребенка, что показано на рис. 2, и поиск его детей в массиве, т.к. у первого

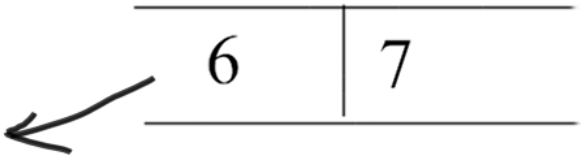

Рисунок 2 — очередь, после «выхода» первого ребенка

ребенка нет детей, то никто из внуков не будет записан. Далее идет проверка второго ребенка, у него есть дети, их индексы записываются в очередь, как это показано на рис. 3. Далее программа продолжает свои действия пока не

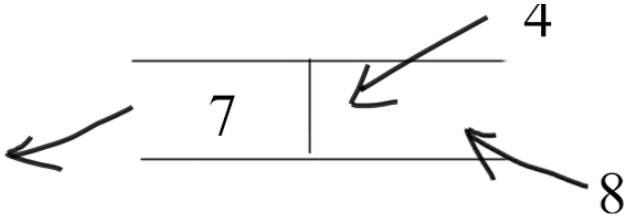

Рисунок 3 — очередь, после «выхода» второго ребенка

очередь будет пуста, данные шаги приведены на рис. 4 и рис. 5.

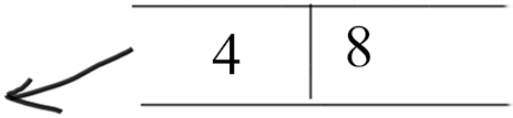

Рисунок 4 — очередь, после «выхода» первого внука

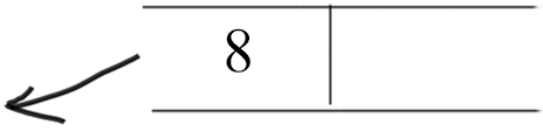

Рисунок 5 — очередь, после «выхода» второго внука

3. ТЕСТИРОВАНИЕ

Программа была собрана в компиляторе G++ в OS Linux Ubuntu 12.04. Программа может быть скомпилирована с помощью команды:

g++ <имя файла>.cpp

Тестовые случаи представлены в Приложении А.

Исходя из тестовых случаев можно увидеть, что в восьмом тесте программа ведет себя некорректно, поэтому была исправлена функция main(): была добавлена проверка на введенное имя. Если введенное имя не из списка имен, то выводится ошибка о неверности введенного имени. Вывод исправлена программы приведен в последующих тестовых случаях. После тестовые случаи не выявили некорректной работы программы , что говорит о том, что по результатам тестирования было показано: поставленная задача была выполнена.

4. ВЫВОД

В ходе выполнения лабораторной работы было выполнено ознакомление с часто используемой на практике линейной структурой данных, обеспечивающей доступ к элементам последовательности только через её начало, и способами реализации этой структуры, было освоено на практике использование очереди для решения задач на языке C++.

Была реализована программа записывающая в файл имена всех потомков введенного пользователем имени в следующем порядке: сначала имена всех его детей, затем всех его внуков, затем всех его правнуков и т.д.

ПРИЛОЖЕНИЕ А. ТЕСТОВЫЕ СЛУЧАИ

№	Ввод	Вывод
1	names1.txt: 5 Alice Sam 1 Bella Charlie 1 Charlie Alice 1 Bella Daniel 1 Input name: Bella	Alice Sam Bella Charlie Daniel 01000 00000 00011 10000 00000 generatin.txt: Charlie Daniel Alice Sam
2	names2.txt: 5 Anna Gleb 1 Boris Kirill 1 Ilya Boris 1 Input name: Anna	Anna Gleb Boris Kirill Ilya 01000 00000 00010 00000 00100 generation.txt: Gleb

3	names3.txt: 7 Garold Bell 1 Garold Ginger 1 Bell Kate 1 Ginger Edgar 1 Kate Tompson 1 Edgar Tompson 1 Bell Arnold 1 Input name: Bell	Garold Bell Ginger Kate Edgar Tompson Arnold 0110000 0001001 0000100 0000010 0000010 0000000 0000000 generation.txt: Kate Arnold Tompson

4	names4.txt: 9 Amelia Bethany 1 Amelia Charlotte 1 Charlotte David 1 Charlotte Evian 1 Hamilton Mia 1 George Evian 1 George Sophia 1 Hamilton George 1 Input name: Amelia	Amelia Bethany Charlotte David Evian Hamilton Mia George Sophia 011000000 000000000 000110000 000000000 000000000 000000110 000000000 000010001 000000000 generation.txt: Bethany Charlotte David Evian
5	names2.txt: 5 Anna Gleb 1 Boris Kirill 1 Ilya Boris 1 Input name: Ilya	Anna Gleb Boris Kirill Ilya 01000 00000 00010 00000 00100 generation.txt: Boris Kirill

6	names3.txt: 7 Garold Bell 1 Garold Ginger 1 Bell Kate 1 Ginger Edgar 1 Kate Tompson 1 Edgar Tompson 1 Bell Arnold 1 Input name: Garold	Garold Bell Ginger Kate Edgar Tompson Arnold 0110000 0001001 0000100 0000010 0000010 0000000 0000000 generation.txt: Bell Ginger Kate Arnold Edgar Tompson
7		Wrong file name.
8	names4.txt: 9 Amelia Bethany 1 Amelia Charlotte 1 Charlotte David 1 Charlotte Evian 1 Hamilton Mia 1 George Evian 1 George Sophia 1 Hamilton George 1 Input name: Antony	

9	names4.txt: 9 Amelia Bethany 1 Amelia Charlotte 1 Charlotte David 1 Charlotte Evian 1 Hamilton Mia 1 George Evian 1 George Sophia 1 Hamilton George 1 Input name: Antony	Wrong name was inputed.
10	names4.txt: 9 Amelia Bethany 1 Amelia Charlotte 1 Charlotte David 1 Charlotte Evian 1 Hamilton Mia 1 George Evian 1 George Sophia 1 Hamilton George 1 Input name:	Wrong name was inputed.

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД ПРОГРАММЫ

main.cpp:

```
#include <iostream>
#include <cstring>
#include <cstdlib>
#include <fstream>
#include <cctype>

#define SIZE_OF_NAME 30

using namespace std;

typedef struct queue{
   int que[SIZE_OF_NAME];
   int startptr,endptr;
}queue;

char** list_of_names(FILE* ptr,int num,bool** children){
   char **name=(char**)calloc(num,sizeof(char*));
   int j=0;
   int i=0;
   name[i]=(char*)calloc(SIZE_OF_NAME,sizeof(char));
   char* gap_name;
   while(!feof(ptr)){
      gap_name=(char*)calloc(SIZE_OF_NAME,sizeof(char));
      fgets(gap_name,SIZE_OF_NAME,ptr);
      gap_name[strlen(gap_name)-1]='\0';
      for(int j=0;j<strlen(gap_name);j++){
         if(isalpha(gap_name[j])){
            name[i][strlen(name[i])]=gap_name[j];
         }else if(gap_name[j]==' '){
            name[i][strlen(name[i])]='\0';
            int counter=0;
            for(int k=0;k<i;k++){
               if(strcmp(name[i],name[k])==0){
                  counter++;
               }
            }
            if(counter==0){
               i++;
               name[i]=(char*)calloc(SIZE_OF_NAME,sizeof(char));
```

```
            }else{
                free(name[i]);
                name[i]=(char*)calloc(SIZE_OF_NAME,sizeof(char));
            }
        }else if(isdigit(gap_name[j])){
            for(int n=0;n<i;n++){
                if(strstr(gap_name,name[n])==gap_name){
                    for(int l=0;l<i;l++){
                        if(strstr(gap_name,name[l])==(gap_name+1+strlen(name[n])) &&
gap_name[j]=='1'){
                            children[n][l]=true;
                        }
                    }
                }
            }
        }
        free(gap_name);
    }
    return name;
}

int isempty(queue *q){
    if(q->endptr < q->startptr)
        return 1;
    else
        return 0;
}

queue* init(){
    queue* q=(queue*)malloc(sizeof(queue));
    q->startptr=1;
    q->endptr=0;
    return q;
}

void push(queue* q,int x){
    if(q->endptr < SIZE_OF_NAME-1){
        if(q->que[q->endptr]!=x){
            q->endptr++;
            q->que[q->endptr]=x;
        }
```

```
    }else{

    }
    return;
}

void pop(queue* q,char** name,FILE* exit){
    if(isempty(q)==1){
        cout<<"Queue is empty!"<<endl;
        return;
    }else{
        fprintf(exit,"%s\n",name[q->que[q->startptr]]);
        for(int i=q->startptr;i<q->endptr;i++)
            q->que[i]=q->que[i+1];
        q->endptr--;
        return;
    }
}

int main(){
    char fname[SIZE_OF_NAME];
    printf("Choose file\n");
    fgets(fname,SIZE_OF_NAME,stdin);
    fname[strlen(fname)-1]='\0';
    FILE * ptr = fopen (fname, "r");
    FILE * exit = fopen("generation.txt", "w");
    if(ptr==NULL){
        printf("Error.\n");
        return 0;
    }
    int num;
    fscanf(ptr,"%d\n",&num);
    bool** children=(bool**)calloc(num,sizeof(bool*));
    for(int i=0;i<num;i++){
        children[i]=(bool*)calloc(num,sizeof(bool));
        for(int j=0;j<num;j++){
            children[i][j]=false;
        }
    }
    char** name=list_of_names(ptr,num,children);
    for(int i=0;i<num;i++)
        cout<<name[i]<<endl;
```

```
    for(int i=0;i<num;i++){
        for(int j=0;j<num;j++)
            cout<<children[i][j];
        cout<<endl;
    }
    fclose(ptr);
    queue* generation=init();
    cout<<"Input name: "<<endl;
    char*person=(char*)calloc(SIZE_OF_NAME,sizeof(char));
    fgets(person,SIZE_OF_NAME,stdin);
    person[strlen(person)-1]='\0';
    for(int i=0;i<num;i++){
        if(strcmp(person,name[i])==0){
            for(int j=0;j<num;j++)
                if(children[i][j]==true)
                    push(generation,j);
        }
    }
    if(isempty(generation)){
        cerr<<"Wrong name was inputed."<<endl;
        return 0;
    }
    while(isempty(generation)!=1){
        for(int i=0;i<num;i++){
            if(children[generation->que[generation->startptr]][i]==true)
                push(generation,i);
        }
        pop(generation,name,exit);
    }
    for(int j=0;j<num;j++){
        free(children[j]);
    }
    free(children);
    free(person);
    fclose(exit);
    return 0;
}
```