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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: Бинарные деревья

Студент гр. 7381 Лукашев Р.С.

Преподаватель Фирсов М. А.

Санкт-Петербург

2018

Задание.

4-в) Задано бинарное дерево b типа BT с произвольным типом элементов. Используя очередь и операции над ней, напечатать все элементы дерева b по уровням: сначала - из корня дерева, затем (слева направо) - из узлов, сыновних по отношению к корню, затем (также слева направо) - из узлов, сыновних по отношению к этим узлам, и т. д. В работе необходимо использовать шаблонное бинарное дерево, реализованное на основе массива. Достаточно использовать один тип для демонстрации работы программы – например, char.

Пояснение задания.

На вход программе подаётся строка, являющаяся скобочной записью бинарного дерева. Необходимо считать дерево в структуру, после чего записать в очередь узлы данного дерева при обходе в ширину, и вывести их на экран.

Описание алгоритма.

Считывается входная строка (максимум – 5000 символов), после чего происходит запись в структуру бинарного дерева. Если при анализе строки выясняется, что строка не является корректной скобочной записью бинарного дерева, выводится сообщение об ошибке и алгоритм завершает работу. Иначе начинается обход в ширину данного дерева. Используется итеративный метод с использованием очереди. В очередь записываются значения узлов при обходе, после чего они выводятся в поток вывода и алгоритм завершает работу.

Описание функций

Название	Выходной параметр	Входные данные	Описание
`walk_wide`	`void`	`btree tree, queue<base> *nodes`	Записывает в очередь nodes узлы дерева tree при обходе его в ширину.

displayFileContents	void	std::istream* in – указатель на входной поток.	Печатает все символы из входного потока in.
get_index	int	void	Возвращает значение индекса последнего считанного символа входной строки.
read_el	base	base* str	Считывает из входной строки str элемент типа base по адресу(base == char) index, и возвращает его.
read_btree	bintree	base* str, btree tree	Считывает из входной строки str бинарное дерево tree, представленное в строке в виде скобочной записи.
error_handler	void	void	Выводит сообщение об ошибке считывания и завершает работу программы.

Описание структур данных

Название	Поле	Описание
template <class base> class queue	struct node	Пара переменных base el и node* prev. Узел очереди, el – значение узла, prev – указатель на предыдущий элемент очереди (следующий в очереди).
	void push()	Добавляет элемент в очередь.
	base pop()	Возвращает значение элемента, находящегося первым в очереди, и удаляет его.
	bool is_null()	Возвращает true, если очередь пуста.
	node* first	Указатель на первый элемент очереди.
	node* last	Указатель на последний элемент очереди.

template <class base> class binary_tree	const adr max_adr = 1000	Максимально возможный адрес узла дерева в массиве mem
	struct node	Узел дерева. Значение узла base info и двойка адресов левого и правого поддеревьев bintree left и bintree right в массиве памяти mem
	node* mem	Массив памяти. Реализована ссылочная адресация памяти, в качестве массива свободных узлов используется левое поле node. 0-адрес считается указателем на NULL, адрес первого элемента (первого записанного массива) – 2.
	bool is_null(bintree t)	Возвращает true, если t – адрес нуля (ссылка на NULL, т.е. пустое поддерево)
	base root(bintree t)	Возвращает значение узла по адресу t
	bintree left(bintree t)	Возвращает адрес левого поддерева узла по адресу t
	bintree right(bintree t)	Возвращает адрес правого поддерева узла по адресу t
	void insert_children(bintree parent, bintree left, bintree right)	Присваивает значения left и right соответствующим полям узла дерева по адресу parent
	bintree create_node(base e)	Создает в памяти mem узел со значением e, сдвигая массив свободной памяти.

Тестирование.

Результатом работы программы является последовательность узлов дерева (тесты 1, 2 и 3), либо сообщение о некорректном вводе (тесты 4, 5).

№ теста	Исходное выражение:	Результат:
1	(a)	push a result: a
2	(a(b)(c))	push a push b push c

		result: abc
3	(a(b(#)(d))(c(e(f)(g))(#)))	push a push b push c push d push e push f push g result: abcdefg
4	(a(b)(c))d	!-excess characters
5	(a # (b))	Input error

Тест 2.

Считали строку “(a(b)(c))” и отправили ее на обработку read_btree. В функции записи бинарного дерева в структуру началась обработка строки. Считан ‘(‘, функцией get_el считан ‘a’, создан узел дерева, после чего последовательно дважды вызваны копии read_btree, для считывания правого и левого поддеревьев. В них созданы и возвращены узлы со значениями ‘b’ и ‘c’ (правые и левые поддеревья этих узлов при этом стали 0), после чего узлу ‘a’ были присвоены адресы правого и левого поддеревьев узлов ‘b’ и ‘c’ соответственно, и функция завершила свою работу. Затем запустился обход в ширину получившегося дерева, были последовательно записаны в очередь символы ‘a’, ‘b’, ‘c’, после чего произошел их вывод на экран, и программа завершила свою работу.

Вывод.

В процессе выполнения лабораторной работы были продуманы, созданы и реализованы на практике алгоритмы и методы работы с бинарным деревом. Бинарное дерево реализовано на базе вектора (ссылочная адресация памяти). С помощью созданного бинарного дерева была написана программа, выводящая в поток вывода последовательность узлов бинарного дерева при обходе его в ширину. Проект создан на языке C++.

Приложение 1. Код программы (lab4.cpp).

```
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <cstring>
#include "bintree.h"
#include "queue.h"

std::ofstream out("output.txt", std::ofstream::out);
std::ifstream input;

void walk_wide(btree tree, queue<base> *nodes) {
            bintree p;
            queue<bintree> q;
            //index of the first entry (tree root) is 2, empty tree is not
being processed
            q.push(2);
            while (!q.is_null()) {
                  p = q.pop();
                  nodes->push(tree.root(p));
                  out << "push " << tree.root(p) << std::endl;
                  std::cout << "push " << tree.root(p) << std::endl;
                  if (!tree.is_null(tree.left(p))) q.push(tree.left(p));
                  if (!tree.is_null(tree.right(p))) q.push(tree.right(p));
            }
}

void displayFileContents(std::istream* in) {
      std::cout << "File contents:\n";
      out << "File contents:\n";
      std::cout << "-begin-\n";
      out << "-begin-\n";
      char c;
      c = in->get();
      while (!in->eof()) {
            std::cout << c;
            out << c;
            c = in->get();
      }
      std::cout << "\n-end-\n";
      out << "\n-end-\n";
}


int main(int argc, char* argv[]){
      btree tree = btree();
      queue<base>* nodes = new queue<base>();
```

```
            base* str = new base[5000];

            if (argc < 2) {
                  std::cout << "Please, enter input string (max 5000 characters):"
<< std::endl;
                  std::cin.getline(str, 5000);
            }
            else {
                  input.open(argv[1], std::ifstream::in);
                  displayFileContents(&input);
                  input.clear();
                  input.seekg(0);
                  input.getline(str, 5000);
                  if (!input.eof()) {
                        out << "!-excess characters";
                        std::cout << "!-excess characters";
                        return 0;
                  }
            }

            read_btree(str, tree);

            if (get_index() < (int)strlen(str)) {
                  out << "!-excess characters";
                  std::cout << "!-excess characters";
                  exit(1);
            }
            walk_wide(tree, nodes);

            out << "result: \n";
            std::cout << "result: \n";
            while (!nodes->is_null()) {
                  base el;
                  el = nodes->pop();
                  out << el;
                  std::cout << el;
            }
            std::cout << std::endl;

            //for (int i = 1; i < 12; i++) {
            //    std::cout << "i:" << i << ' ' << tree.mem[i].left << ' ' <<
tree.mem[i].info << ' ' << tree.mem[i].right << std::endl;
            //}
            delete nodes;
            delete [] str;
            std::cin.get();
          return 0;
      }
```

Приложение 2. Реализация очереди (queue.h).

```
#pragma once
#include <stdlib.h>
//shortened queue class
template <class base> class queue {
private:
	struct node {
		base el;
		node* prev;
		node() {
			prev = NULL;
		}
	};
	node* first = NULL;
	node* last = NULL;
public:

	queue() {
		first = NULL;
		last = NULL;
	}

	bool is_null() {
		return !first;
	}

	void push(base element) {
		node* p = new node();
		p->el = element;
		if (first == NULL) {
			last = p;
			first = p;
			return;
		}
		last->prev = p;
		last = p;
	}

	base pop() {
		base res;
		if (first) res = first->el;
		node* prev;
		prev = first->prev;
		delete first;
		first = prev;
		return res;
	}
```

```
};
```

Приложение 3. Содержимое файла queue.cpp.

```
#include "queue.h"
```

Приложение 4. Реализация бинарного дерева (bintree.h).

```
#pragma once
#include <stdlib.h>
#include <cstring>
template <class base> class binary_tree{
private:

public:
	typedef size_t adr;
	const adr max_adr = 1000;
	//bintree == index of node
	typedef adr bintree;

	struct node {
		base info;
		bintree left;
		bintree right;
	};

	node* mem;

	binary_tree() {
		mem = new node[max_adr + 1];
		for (int i = 0; i <= max_adr; i++) {
			mem[i].left = (i+1) % max_adr;
			mem[i].right = 0;
		}
	}

	bool is_null(bintree t) {
		//index 0 is NULL pointer
		return !t;
	}

	base root(bintree t) {
		if (t > max_adr) return 0;
		return mem[t].info;
	}

	bintree left(bintree t) {
		if (t > max_adr) return 0;
		return mem[t].left;
```

```
	}

	bintree right(bintree t) {
		if (t > max_adr) return 0;
		return mem[t].right;
	}

	void insert_children(bintree parent, bintree left, bintree right) {
		if (parent > max_adr || left > max_adr || right > max_adr) return;
		mem[parent].left = left;
		mem[parent].right = right;
	}

	bintree create_node(base e) {
		if (mem[1].left == 0) return 0;
		node p;
		p.info = e;
		p.right = 0;
		p.left = 0;
		bintree freemem = mem[mem[1].left].left;
		mem[mem[1].left] = p;
		bintree res = mem[1].left;
		mem[1].left = freemem;
		return res;
	}

};

//char btree functions and type defenitions
typedef char base;
typedef binary_tree<base> btree;
typedef btree::bintree bintree;
typedef binary_tree<base>::node node;
bintree read_btree(base* str, btree tree);
int get_index();
void error_handler();
```

Приложение 5. Реализация вспомогательных функций для бинарного дерева char'ов (bintree.cpp).

```
#include "bintree.h"
#include <fstream>
#include <stdlib.h>
#include <iostream>

int index_ = 0;
std::ofstream inter_data("interData.txt", std::ofstream::out);

int get_index(){
```

```
	return index_;
}
//base == char
base read_el(base* str) {
	base el;
	while (*(str + index_) == ' ') index_++;
	el = *(str + index_);
	index_++;
	return el;
}
//example (a(b(#)(d))(c(e(f)(g))(#)))
bintree read_btree(base* str, btree tree) {

	bintree res = 0;
	if (index_ > strlen(str)) error_handler();
	while (*(str + index_) == ' ') index_++;
	if (*(str + index_) == ')') {

		return 0;
	}
	if (*(str + index_) == '(') {

		index_++;
		base el = read_el(str);
		if (el == ')' || el == '(') error_handler();
		if (el == '#') {

			inter_data << "met empty leaf" << std::endl;

		}
		else {
			res = tree.create_node(el);

			inter_data << "create node " << el << std::endl;

			bintree left = read_btree(str, tree);
			while (*(str + index_) == ' ') index_++;
			bintree right = read_btree(str, tree);
			while (*(str + index_) == ' ') index_++;
			tree.insert_children(res, left, right);
			inter_data << "cons " << tree.root(res) << ' ' <<
tree.root(left) << ' ' << tree.root(right) << std::endl;
		}
	}
	if (*(str + index_) != ')') error_handler();

	index_++;
	return res;
}
```

```
	void error_handler() {
		std::cout << "Input error" << std::endl;
		std::cin.get();
		exit(1);
}
```

Приложение 6. Содержимое файла Makefile.

```
CODE  = ./Source/
OBJ   = lab4.o queue.o bintree.o
EXE   = lab3
CXX         = g++
CFLAGS      = -Wall -Wextra -fpermissive -c -static

all:  $(OBJ)
	$(CXX) $(OBJ) -o $(EXE)

lab4.o: $(CODE)lab4.cpp
	$(CXX) $(CFLAGS) $(CODE)lab4.cpp

queue.o: $(CODE)queue.cpp $(CODE)queue.h
	$(CXX) $(CFLAGS) $(CODE)queue.cpp

bintree.o: $(CODE)bintree.cpp $(CODE)bintree.h
	$(CXX) $(CFLAGS) $(CODE)bintree.cpp

clean:
rm $(OBJ) $(EXE)
```

Приложение 5. Содержимое файла checker.sh.

```
#!/bin/bash
touch checker_res.txt
cp /dev/null checker_res.txt
for cfile in Tests/Correct/*; do
    echo "running test: \"Tests/Correct/$cfile\" ";
      echo "correct test $cfile:" >>checker_res.txt;
    ./lab3 $cfile >>checker_res.txt;
      echo " ">>checker_res.txt;
done;
for icfile in Tests/Incorrect/*; do
    echo "running test: \"Tests/Correct/$icfile\" ";
      echo "incorrect test $icfile:" >>checker_res.txt;
    ./lab3 $icfile >>checker_res.txt;
      echo " ">>checker_res.txt;
done;
```