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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Алгоритмы и структуры данных»

Тема: Иерархические списки

Студент гр. 8304 _____ Карабанов Р.Е

Преподаватель _____ Фирсов М.А

Санкт-Петербург

2019

Цель работы.

Ознакомиться с основными понятиями и структурой иерархических списков.

Задание.

Вариант № 7.

удалить из иерархического списка все вхождения заданного элемента (атома) x;

Описание функций краткое описание алгоритма.

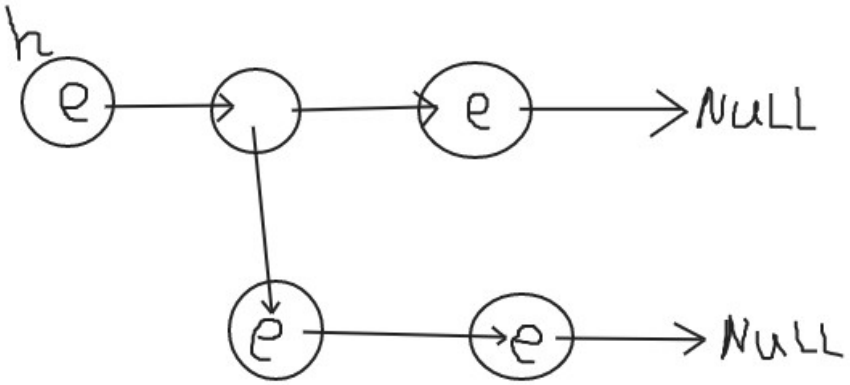
list* createlist(char* str, int* i, list* parent) — функция принимает указатель на строку, указатель на индекс (был выбран во избежание глобальной переменной) элемент списка которым были порождены создаваемые. В функции создаётся элемент - «голова» списка, затем проходя до конца строки проверяется чем является элемент строки, если это открывающая скобка, то рекурсивно вызывается эта же функция, результат которой будет записан в текущий элемент списка, а именно в его «потомство». Если встречается открывающая скобка, то происходит вовращение указателя на начало данного списка. Иначе происходи добавление элемента в список. В конце, для обозначения конца списка указатель следующего элемента у последнего элемента присваивается NULL. Возвращается указатель на «голову» списка.

list* CheckAndDelete(list* head, char elemforrem) - функция принимаем указатель на первый элемент в списке, и символ для удаления. В функции происходит проверка каждого элемента списка на присутствие в его поле elem — элемента для удаления, если такой имеется элемент списка удаляется. Если встречается элемент списка, у которой есть «потомство» (поле child не

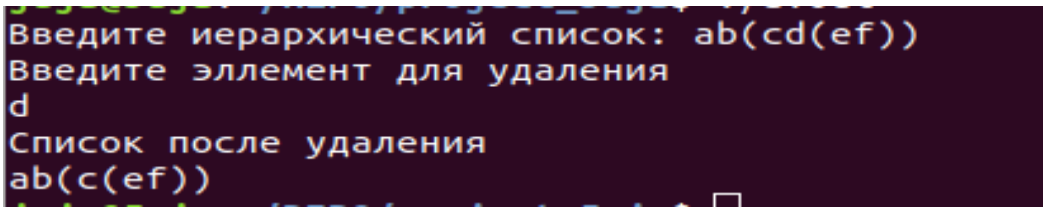
равно NULL), то функция рекурсивно заходит сама в себя. Возвращается указатель на «голову» списка.

void PrintList(list* head) - функция печати обработанного списка, если у рассматриваемого элемента списка есть «потомство» то рекурсивно заходит сама в себя. Ничего не возвращает, прекращает работу после нахождения последнего элемента в текущем списке.

Представление иерархического списка в памяти.



Тестирование.



Ниже представлена таблица № 1, в которой представлены некоторые примеры работы программы.

Таблица №1 примеры работы программы.

Строка	Символ для удаления	Вывод
aaaaaaaaaa	a	
ffff(aaaa)fff	a	fffffff
(fd(dc)(d)g)	d	(f(c)g)

Выводы.

В ходе лабораторной работы были изучены основные понятия и структура иерархических списков.

Исходный код.

```
#include <iostream>
#include <cstring>

//структура для элемента списка
typedef struct list {
	struct list* parent;
	struct list* child;
	struct list* next;
	char elem;
}list;

/*рекурсивная функция создания списка*/
list* createlist(char* str, int* i, list* parent);
//функция удаления заданного элемента из списка
list* CheckAndDelete(list* head, char ellemforrem);
//функция вывода на экра обработанного списка
void PrintList(list* head);

int main(int argc, char* argv[]) {
	char* str = (char*)calloc(200, sizeof(char));
```

```
		char c;
		// если не указана папка с тестами, то пользователь вводи строку
		if (argc == 1){
			printf("Введите иерархический список: ");
			std::cin >> str;
			std::cout << "Введите эллемент для удаления" << std::endl;
			std::cin >> c;
		}
		else {
			char* path = (char*)calloc(strlen(argv[1]) + 8, sizeof(char)); // строка для
открытия файла
			char* estr; // указатель для отслеживания конца файла
			strcat(path, "Tests/");
			strcat(path, argv[1]);
			FILE* fileTests = fopen(path, "r"); // открываем файл
			if (!fileTests){
				printf("Ошибка открытия файла\n");
				return 0;
			}
			estr = fgets(str, 199, fileTests); // считываем строку из файла
			// если указатель NULL то проверяем на ошибку или конец файла
			if (estr == NULL){
				if (feof(fileTests) != 0){
					printf("Конец файла\n");
				}
				else {
					printf("Ошибка чтения файла\n");
				}
			}
			//считываем символ, который необходимо удалить из списка
			c = getc(fileTests);
			free(path);
			fclose(fileTests);
		}
		int i = 0;
		list* Ierlist = createlist(str, &i, NULL);
		Ierlist = CheckAndDelete(Ierlist, c);
		std::cout << "Список после удаления\n";
		PrintList(Ierlist);
```

```
	std::cout << std::endl;
	free(str);
	return 0;
}
/*функция принимает указатель на строку, указатель на индекс
(был выбран во избежание глобальной переменной)
элемент списка которым были порождены создаваемые*/
list* createlist(char* str, int* i, list* parent) {
	list* head = (list*)malloc(sizeof(list));
	list* cur = head;
	//пока не конец строки обрабатываем её
	while (str[*i] != '\0' && str[*i] != '\n') {
// заходим на уровень ниже, создаём порождаемый список
		if (str[*i] == '(') {
			(*i)++;
			cur->child = createlist(str, i, cur);
// для определённости обозначим в "указательном" элементе значение хранящегося символа
			cur->elem = '\0';
		}
// выходим из данного уровня рекурсии
		else if (str[*i] == ')') {
			(*i)++;
			cur->next = NULL;
			return head;
		}
// добавляем элемент в список
		else {
			cur->elem = str[*i];
			cur->child = NULL;
			cur->parent = parent;
			(*i)++;
		}
// доп. проверка для выделения памяти.
		if (str[*i] != '\0' && str[*i] != ')' && str[*i] != '\n') {
			cur->next = (list*)malloc(sizeof(list));
			cur = cur->next;
		}
	}
```

```
// обозначения конца списка
	cur->next = NULL;
// возвращаем указатель на 1й элемент
	return head;
}

/*функция принимаем указатель на первый элемент в списке,
и символ для удаления*/
list* CheckAndDelete(list* head, char ellemforrem) {
// вспомогательный указатель
	list* ptr;
	list* cur = head;
// пока не конец списка
	while (cur != NULL) {
		if (cur->elem == ellemforrem) {
// удаления 1го элемента списка
			if (cur == head) {
				cur = cur->next;
				free(head);
				head = cur;
				continue;
			}
// удаления элемента
			else {
				ptr = head;
				while (ptr->next != cur) {
					ptr = ptr->next;
				}
				ptr->next = cur->next;
				free(cur);
				cur = ptr;
			}
		}
// если от элемент является "указательным"
		else if (cur->child != NULL) {
			cur->child = CheckAndDelete(cur->child, ellemforrem);
		}
		cur = cur->next;
	}
```

```
// возвращаем указатель на 1й элемент
	return head;
}

// выводим список
void PrintList(list* head) {
	list* cur = head;
	while (cur != NULL) {
		if (cur->child != NULL) {
			std::cout << '(';
			PrintList(cur->child);
			std::cout << ')';
			cur = cur->next;
		}
		else {
			std::cout << cur->elem;
			cur = cur->next;
		}
	}
}
```