МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Кафедра «Вычислительная техника»

ОТЧЁТ

По лабораторной работе №3

По курсу «Логика и основы алгоритмизации в инженерных задачах»

На тему «Динамические списки»

Выполнили
студенты
группы
23ВВВ4:
Святов И.Ю.
Епинин Д.В.

Приняли:
Юрова О.В.
Деев М.В.

Пенза 2024

Общие сведения.

Общие сведения:

Список представляет собой последовательность элементов определенного типа. Простейший тип списка – линейный, когда для каждого из элементов, кроме последнего, имеется следующий, и для каждого, кроме первого имеется предыдущий элемент.

Возможна реализация списков посредством массивов или динамическая реализация.

Динамические списки относятся к динамическим структурам и используются, когда размер данных заранее неизвестен. Созданием динамических данных должна заниматься сама программа во время своего исполнения, этим достигается эффективное распределение памяти, но снижается эффективность доступа к элементам.

Динамические структуры данных отличаются от статических двумя основными свойствами:

1) в них нельзя обеспечить хранение в заголовке всей информации о структуре, поэтому каждый элемент должен содержать информацию, логически связывающую его с другими элементами структуры;

2) для них зачастую не удобно использовать единый массив смежных элементов памяти, поэтому необходимо предусматривать ту или иную схему динамического управления памятью.

Для обращения к динамическим данным применяют указатели.

Набор операций над списком будет включать добавление и удаление элементов, поиск элементов списка.

Различают односвязные, двусвязные и циклические списки.

В простейшем случае каждый элемент содержит всего одну ссылку на следующий элемент, такой список называется односвязным.

В простейшем случае для создания элемента списка используется структура, в которой объединяются полезная информация и ссылка на следующий элемент списка:

```
struct node
{
	char inf[256];  // полезная информация
	struct node *next; // ссылка на следующий элемент
};
```

Задание:

1. Реализовать приоритетную очередь, путём добавления элемента в список в соответствии с приоритетом объекта (т.е. объект с большим приоритетом становится перед объектом с меньшим приоритетом).

Ответ:

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream>
#include <string.h>
#include <locale.h>

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct Node {
	int priority; // Приоритет элемента
	char* data; // Данные элемента
	struct Node* next; // Указатель на следующий элемент
} Node;
```

```
typedef struct PriorityQueue {
	Node* head; // Указатель на голову списка
} PriorityQueue;

// Функция создания новой приоритетной очереди
PriorityQueue* create_queue() {
	PriorityQueue* pq = (PriorityQueue*)malloc(sizeof(PriorityQueue));
	pq->head = NULL;
	return pq;
}

// Функция добавления элемента в приоритетную очередь
void enqueue(PriorityQueue* pq, const char* data, int priority) {
	Node* new_node = (Node*)malloc(sizeof(Node));
	new_node->priority = priority;
	new_node->data = _strdup(data); // Копируем строку
	new_node->next = NULL;

	// Если очередь пуста либо приоритет нового элемента выше приоритета головы
	if (pq->head == NULL || pq->head->priority < priority) {
		new_node->next = pq->head;
		pq->head = new_node;
	}
	else {
		// Ищем место для вставки нового элемента
		Node* current = pq->head;
		while (current->next != NULL && current->next->priority >= priority) {
			current = current->next;
		}
		new_node->next = current->next;
		current->next = new_node;
	}
}

// Функция извлечения элемента с наивысшим приоритетом
char* dequeue(PriorityQueue* pq) {
	if (pq->head == NULL) {
		return NULL; // Очередь пуста
	}
	Node* temp = pq->head;
	char* data = _strdup(temp->data); // Копируем данные
	pq->head = pq->head->next;
	free(temp->data); // Освобождаем память для данных
	free(temp); // Освобождаем память для узла
	return data; // Возвращаем данные
}

// Функция для просмотра очереди
void review_queue(PriorityQueue* pq) {
	Node* current = pq->head;
	if (current == NULL) {
		printf("Очередь пуста\n");
		return;
	}

	printf("Содержимое очереди:\n");
	while (current != NULL) {
		printf("Имя - %s, Приоритет - %d\n", current->data, current->priority);
		current = current->next;

	}
}

// Функция проверки, пуста ли очередь
int is_empty(PriorityQueue* pq) {
	return pq->head == NULL;
}

// Функция освобождения памяти очереди
```

```
void free_queue(PriorityQueue* pq) {
	Node* current = pq->head;
	Node* next_node;
	while (current != NULL) {
		next_node = current->next;
		free(current->data); // Освобождаем память для данных
		free(current);
		current = next_node;
	}
	free(pq);
}

// Функция для ввода новой задачи
void enter_task(PriorityQueue* pq) {
	char data[256]; // Массив для хранения названия задачи
	int priority;

	printf("Введите задачу: ");
	getchar(); // Сбрасываем символ новой строки после scanf
	fgets(data, sizeof(data), stdin);
	data[strcspn(data, "\n")] = '\0'; // Удаляем символ новой строки, если он есть
	printf("Введите приоритет (число): ");
	scanf("%d", &priority);

	enqueue(pq, data, priority);
}

// Пример использования
int main() {
	setlocale(LC_ALL, "Russian");
	PriorityQueue* pq = create_queue();
	int choice;

	while (1) {
		printf("\nПожалуйста, выберите действие:\n");
		printf("1. Добавить задачу\n");
		printf("2. Извлечь задачу с наивысшим приоритетом \n");
		printf("3. Просмотр очереди \n");
		printf("4. Выход\n");
		printf("Ваш выбор: ");
		scanf("%d", &choice);

		switch (choice) {
		case 1:
			enter_task(pq);
			break;
		case 2:
		{
			char* task = dequeue(pq);
			if (task) {
				printf("Обработка задачи: %s\n", task);
				free(task); // Освобождение памяти для задач
			}
			else {
				printf("Очередь пуста.\n");
			}
		}
		break;
		case 3:
			review_queue(pq);
			break;
		case 4:
			exit(0);
		default:
			printf("Неверный выбор. Попробуйте снова.\n");
		}
	}
	free_queue(pq); // Освобождаем память для очереди
	return 0;   }
```

2. * На основе приведенного кода реализуйте структуру данных *Очередь*.

Ответ:

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream>
#include <string.h>
#include <locale.h>

struct node {
    char inf[256];
    struct node* next;
};

struct node* head = NULL, * last = NULL;

struct node* get_struct(void);
void enqueue(void);
char* dequeue(void);
void review(void);
int is_empty(void);

struct node* get_struct(void) {
    struct node* p = NULL;

    if ((p = (struct node*)malloc(sizeof(struct node))) == NULL) {
        printf("Ошибка при распределении памяти\n");
        exit(1);
    }

    printf("Введите название объекта: \n");
    scanf("%s", p->inf);

    p->next = NULL;
    return p;
}

/* Добавление элемента в конец очереди */
void enqueue(void) {
    struct node* p = get_struct();

    if (head == NULL) { // Если очередь пуста
        head = p;       // Устанавливаем голову
        last = p;       // Устанавливаем конец
    }
    else {            // Если очередь не пуста
        last->next = p; // Добавляем в конец
        last = p;       // Обновляем указатель на конец
    }
}

/* Извлечение элемента из начала очереди */
char* dequeue(void) {
    if (head == NULL) { // Если очередь пуста
        printf("Очередь пуста\n");
        return NULL;
    }

    struct node* temp = head;  // Сохраняем элемент на удаление
    char* data = _strdup(temp->inf); // Копируем данные для возврата
    head = head->next;        // Обновляем голову очереди
    free(temp);               // Освобождаем память
    return data;              // Возвращаем данные
```

```
}

/* Просмотр содержимого очереди */
void review(void) {
    struct node* struc = head;
    if (head == NULL) {
        printf("Очередь пуста\n");
        return;
    }
    while (struc) {
        printf("Имя - %s\n", struc->inf);
        struc = struc->next;
    }
}

/* Проверка, пуста ли очередь */
int is_empty(void) {
    return head == NULL;
}

/* Пример использования очереди */
int main() {
    setlocale(LC_ALL, "Russian");
    int choice;
    char* data;

    while (1) {
        printf("1. Добавить элемент в очередь\n");
        printf("2. Извлечь элемент из очереди\n");
        printf("3. Просмотр очереди\n");
        printf("4. Выход\n");
        printf("Выберите действие: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            enqueue();
            break;
        case 2:
            data = dequeue();
            if (data) {
                printf("Извлеченный элемент: %s\n", data);
                free(data); // Освобождение памяти для извлеченных данных
            }
            break;
        case 3:
            review();
            break;
        case 4:
            exit(0);
        default:
            printf("Некорректный ввод\n");
            break;
        }
    }
    return 0;
}
```

3. * На основе приведенного кода реализуйте структуру данных *Стек*.
Ответ:

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream>
#include <string.h>
#include <locale.h>

struct node {
    char inf[256];
    struct node* next;
};

struct node* top = NULL; // Указатель на вершину стека

struct node* get_struct(void);
void push(void);
char* pop(void);
void review(void);
int is_empty(void);

struct node* get_struct(void) {
    struct node* p = NULL;

    if ((p = (struct node*)malloc(sizeof(struct node))) == NULL) {
        printf("Ошибка при распределении памяти\n");
        exit(1);
    }

    printf("Введите название объекта: \n");
    scanf("%s", p->inf);

    p->next = NULL;
    return p;
}

/* Добавление элемента в стек */
void push(void) {
    struct node* p = get_struct();

    if (top == NULL) { // Если стек пуст
        top = p; // Устанавливаем вершину на новый элемент
    }
    else {
        p->next = top; // Новый элемент указывает на старую вершину
        top = p; // Обновляем вершину стека
    }
}

/* Извлечение элемента из стека */
char* pop(void) {
    if (top == NULL) { // Если стек пуст
        printf("Стек пуст\n");
        return NULL;
    }

    struct node* temp = top; // Сохраняем вершину для удаления
    char* data = _strdup(temp->inf); // Копируем данные для возврата
    top = top->next; // Обновляем вершину стека
    free(temp); // Освобождаем память
    return data; // Возвращаем данные
}

/* Просмотр содержимого стека */
void review(void) {
```

```
    struct node* struc = top;
    if (top == NULL) {
        printf("Стек пуст\n");
        return;
    }
    while (struc) {
        printf("Имя - %s\n", struc->inf);
        struc = struc->next;
    }
}

/* Проверка, пуст ли стек */
int is_empty(void) {
    return top == NULL;
}

/* Пример использования стека */
int main() {
    setlocale(LC_ALL, "Russian");
    int choice;
    char* data;

    while (1) {
        printf("1. Добавить элемент в стек\n");
        printf("2. Извлечь элемент из стека\n");
        printf("3. Просмотр стека\n");
        printf("4. Выход\n");
        printf("Выберите действие: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            push();
            break;
        case 2:
            data = pop();
            if (data) {
                printf("Извлеченный элемент: %s\n", data);
                free(data); // Освобождаем память для извлеченных данных
            }
            break;
        case 3:
            review();
            break;
        case 4:
            exit(0);
        default:
            printf("Некорректный ввод\n");
            break;
        }
    }
    return 0;
}
```

Вывод: В ходе лабораторной работы научился представлять список в последовательность элементов определенного типа. Сделал простейший тип списка – линейный, а также изучил принцип работы динамического списка.