

Министерство науки и высшего образования РФ
Пензенский государственный университет
Кафедра “Вычислительная техника”

Отчёт

по лабораторной работе №3
по курсу “Логика и основы алгоритмизации в инженерных задачах”
на тему “Динамические списки”

Выполнил студент гр. 22ВВВ3:
Кулахметов С.И.

Приняли:
к.т.н., доцент Юрова О.В.
к.э.н., доцент Акифьев И.В.

Пенза 2023

Цель работы

Рассмотреть принципы работы динамических структур (односвязных списков), научиться реализовывать их на языке программирования Си. Выполнить лабораторное задание.

Лабораторное 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).

2. Реализовать структуру данных *Очередь*.

3. Реализовать структуру данных *Стек*.

Пояснительный текст к программам

Лабораторное задание выполнено в виде трёх программ, каждая из которых отвечает за реализацию одной структуры данных.

В первой программе реализована структура данных *Приоритетная очередь*, которая, как и обычная очередь придерживается принципа FIFO, однако, данные извлекаются из неё в порядке наибольшего приоритета. Реализация данного алгоритма основана на односвязном списке, в основе которого лежит следующая структура данных:

```
struct Queue
{
    int data;    // Числовой элемент очереди
    int priority; // Приоритет элемента
    struct Queue *next; // Указатель на следующий элемент очереди
};
```

Во второй программе представлена реализация простой очереди, написанной по аналогии с первой программой, но без учёта приоритета при извлечении данных из структуры. Добавление происходит в конец списка, а чтение с удалением элемента — с начала.

Третья программа отвечает за реализацию структуры данных *Стек*, который придерживается принципа FILO. Добавление и чтение с удалением элемента происходят с конца списка.

Так как все алгоритмы реализованы через динамические списки, то после ввода элемента выводится запрос на повторение внесения данных в структуру.

Результаты работы программ

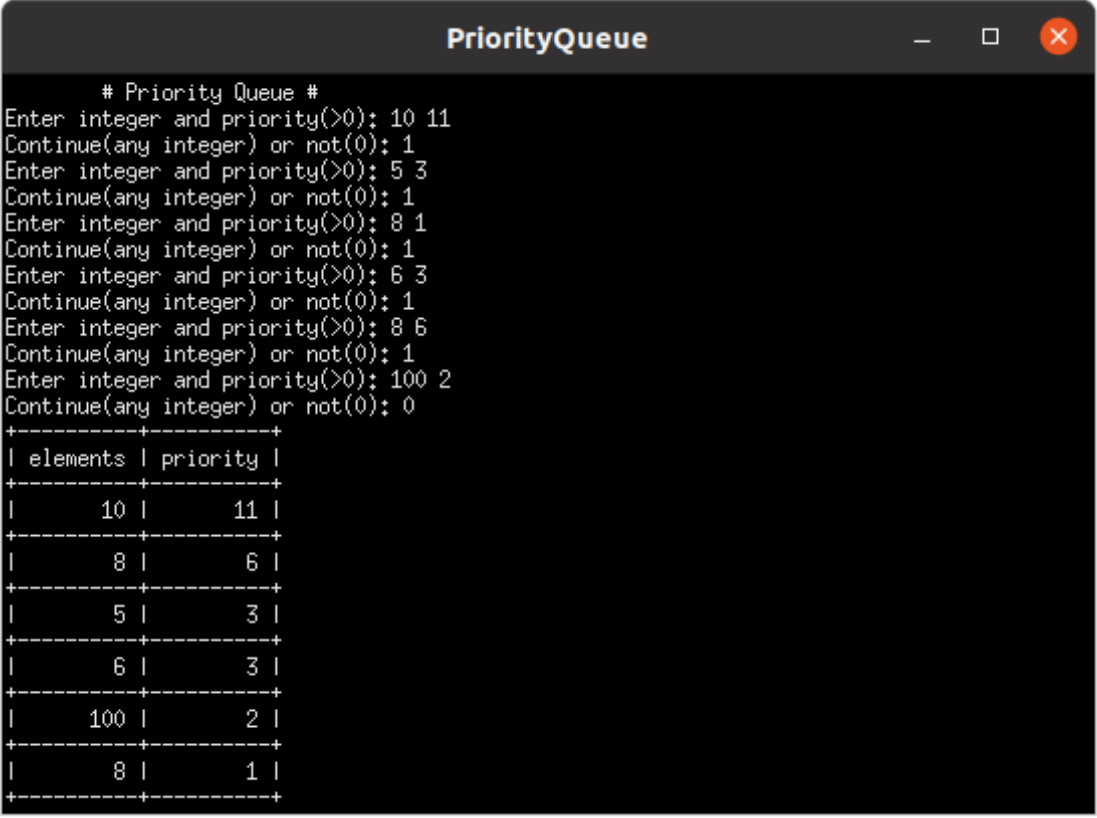
Задание 1

Данные заносятся в приоритетную очередь в следующем порядке:

- 1) 10 (приоритет 11)
- 2) 5 (приоритет 3)
- 3) 8 (приоритет 1)
- 4) 6 (приоритет 3)
- 5) 8 (приоритет 6)
- 6) 100 (приоритет 2)

(см. Приложение Б рис. 1)

Результат извлечения данных из приоритетной очереди.



```
# PriorityQueue #
Enter integer and priority(>0): 10 11
Continue(any integer) or not(0): 1
Enter integer and priority(>0): 5 3
Continue(any integer) or not(0): 1
Enter integer and priority(>0): 8 1
Continue(any integer) or not(0): 1
Enter integer and priority(>0): 6 3
Continue(any integer) or not(0): 1
Enter integer and priority(>0): 8 6
Continue(any integer) or not(0): 1
Enter integer and priority(>0): 100 2
Continue(any integer) or not(0): 0
+-----+-----+
| elements | priority |
+-----+-----+
|      10 |      11 |
+-----+-----+
|       8 |       6 |
+-----+-----+
|       5 |       3 |
+-----+-----+
|       6 |       3 |
+-----+-----+
|     100 |       2 |
+-----+-----+
|       8 |       1 |
+-----+-----+
```

Стоит обратить внимание на то, что если приоритет данных одинаковый, то они извлекаются в стандартном порядке обычной очереди.

Задание 2

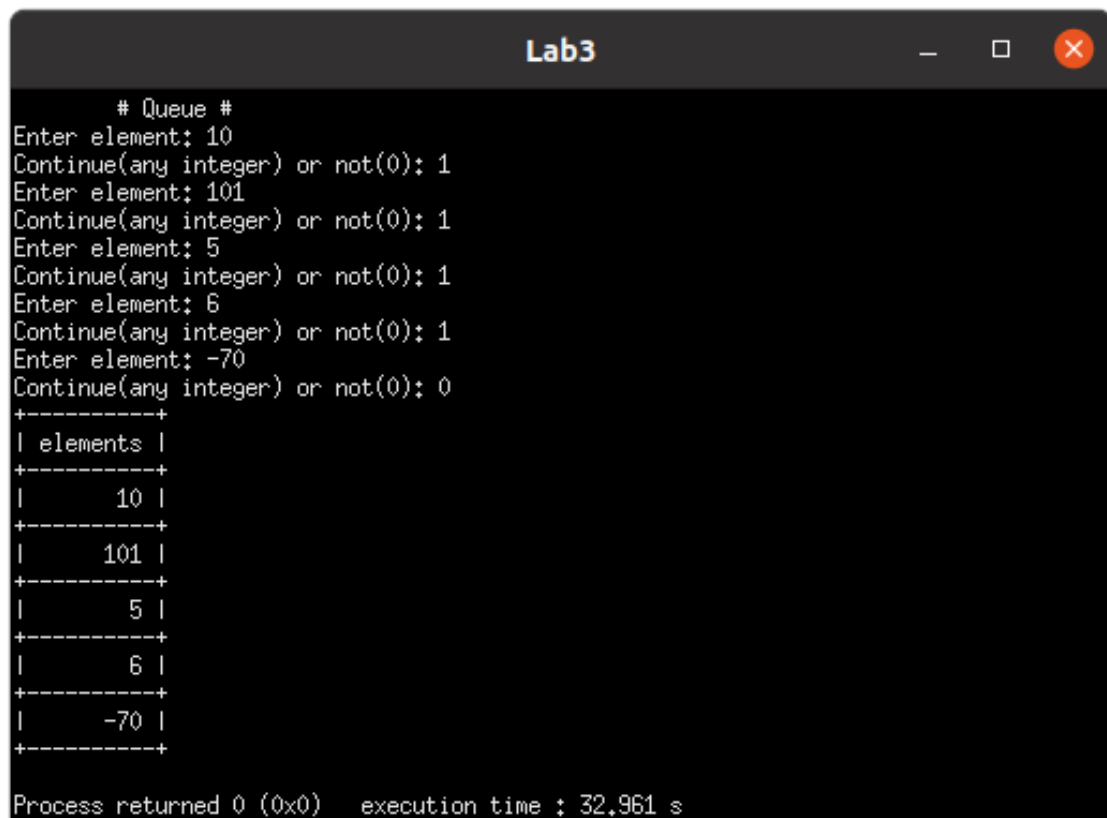
Данные заносятся в очередь в следующем порядке:

- 1) 10
- 2) 101

- 3) 5
- 4) 6
- 5) -70

(см. Приложение Б рис. 2)

Результат извлечения данных из очереди.



```
# Queue #
Enter element: 10
Continue(any integer) or not(0): 1
Enter element: 101
Continue(any integer) or not(0): 1
Enter element: 5
Continue(any integer) or not(0): 1
Enter element: 6
Continue(any integer) or not(0): 1
Enter element: -70
Continue(any integer) or not(0): 0
+-----+
| elements |
+-----+
|      10 |
+-----+
|     101 |
+-----+
|       5 |
+-----+
|       6 |
+-----+
|     -70 |
+-----+

Process returned 0 (0x0)   execution time : 32.961 s
```

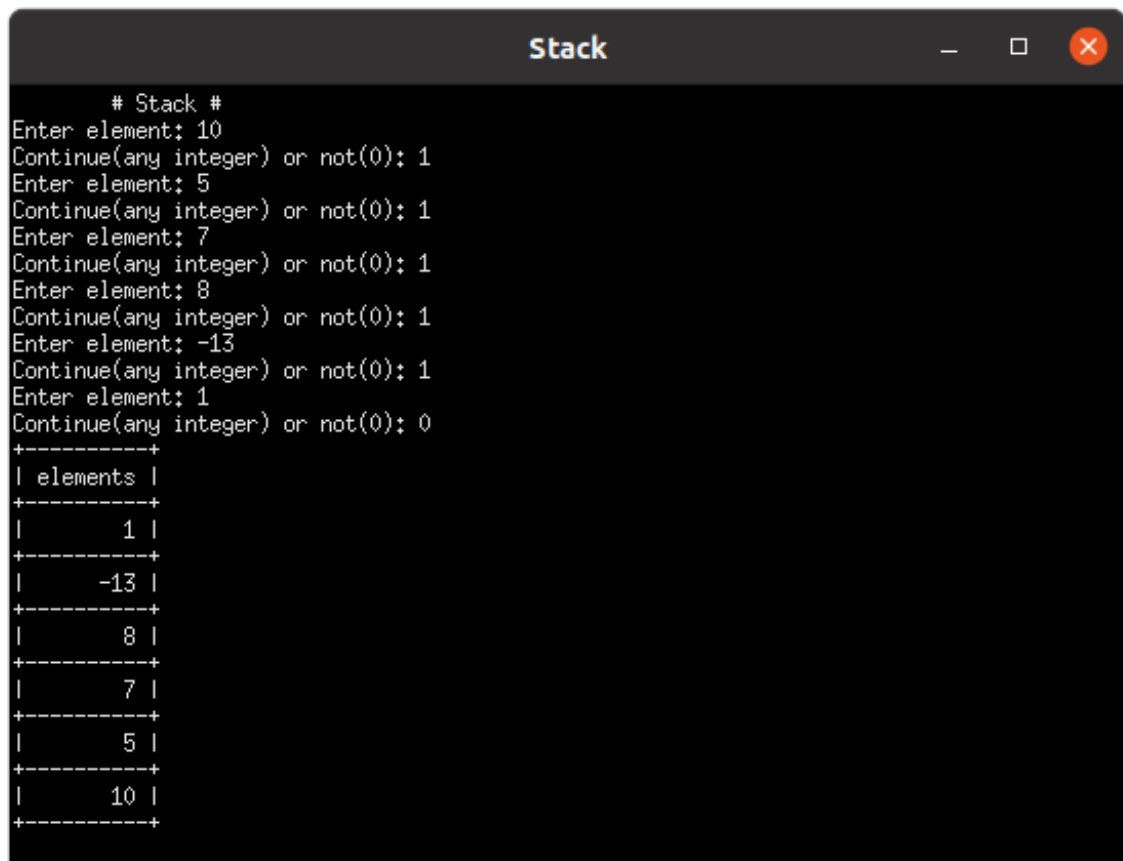
Задание 3

Данные заносятся в стек в следующем порядке:

- 1) 10
- 2) 5
- 3) 7
- 4) 8
- 5) -13
- 6) 1

(см. Приложение Б рис. 3)

Результат извлечения данных из стека.



```
# Stack #
Enter element: 10
Continue(any integer) or not(0): 1
Enter element: 5
Continue(any integer) or not(0): 1
Enter element: 7
Continue(any integer) or not(0): 1
Enter element: 8
Continue(any integer) or not(0): 1
Enter element: -13
Continue(any integer) or not(0): 1
Enter element: 1
Continue(any integer) or not(0): 0
+-----+
| elements |
+-----+
|      1 |
+-----+
|     -13 |
+-----+
|      8 |
+-----+
|      7 |
+-----+
|      5 |
+-----+
|     10 |
+-----+
```

Вывод

В ходе выполнения данной лабораторной работы были получены навыки реализации динамических списков и основанных на них структур данных. На языке Си написаны: *Приоритетная очередь*, *Очередь* и *Стек*. Проверена их работоспособность и учтён нюанс работы приоритетной очереди — данные с одинаковым приоритетом извлекаются в стандартном порядке очереди.

Ссылка на *GitHub* репозиторий с лабораторной работой

<https://github.com/KulakhmetovS/Lab3>

Приложение А

Листинг программ

Файл PriorityQueue/main.c

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

struct Queue; // Структура, отвечающая за элементы очереди
struct Queue *init(int, int); // Инициализация очереди
void Push(struct Queue **, int, int); // Добавление элемента в очередь
struct Queue *Priority(struct Queue *); // Сортировка элементов в
соответствии с их приоритетом
struct Queue *QueueSort(struct Queue *, int); //Создание отсортированной
очереди
struct Queue *Pop(struct Queue *); // Чтение элемента из очереди с
последующим удалением

int res = 0; // Результат извлечения из очереди
int pr = 0; // Результат извлечения приоритета
int *array; // Указатель на массив с приоритетами

int main()
{
    int element = 0, prio = 0, cont = 0;
    struct Queue *list = NULL; // Указатель на начало очереди

    printf("\t# Priority Queue #\n");
    init: // Метка на случай, если приоритет <= 0
    printf("Enter integer and priority(>0): ");
    scanf("%d%d", &element, &prio);
    if(prio > 0)
    {
        list = init(element, prio); // Инициализация очереди
    }
    else
    {
        printf("priority must be more than 0\n");
        goto init;
    }

    printf("Continue(any integer) or not(0): ");
    scanf("%d", &cont);
    if(cont == 0) goto printing;

    while(cont != 0)
    {
        entering: // Метка на случай, если приоритет <= 0
        printf("Enter integer and priority(>0): ");
        scanf("%d%d", &element, &prio);

        if(prio > 0)
        {
            Push(&list, element, prio);
        }
        else
        {
            printf("priority must be more than 0\n");
            goto entering;
        }

        printf("Continue(any integer) or not(0): ");
        scanf("%d", &cont);
        if(cont == 0) break;
    }
}
```

```

        // Вывод отсортированной очереди
        printf("+-----+-----+\n| elements | priority |\n+-----+-----+\n");
        while(list != NULL)
        {
            list = Pop(list);
            printf("|%9d |%9d |\n+-----+-----+\n", res, pr);
        }

        goto ret;

    printing:
        printf("+-----+-----+\n| element | priority |\n+-----+-----+\n");
        Pop(list);
        printf("|%9d |%9d |\n+-----+-----+\n", res, pr);

    ret:
        return 0;
}

struct Queue
{
    int data; // Числовой элемент очереди
    int priority; // Приоритет очереди
    struct Queue *next; // Указатель на следующий элемент очереди
};

struct Queue *init(int element, int prio)
{
    struct Queue *p = NULL; // Создание указателя на структуру

    p = malloc(sizeof(struct Queue)); // Выделение памяти под структуру

    p -> data = element; // Присваивание введённого значения полю данных
    p -> priority = prio; // Присваивание введённого приоритета
    p -> next = NULL; // Установка на нулевой указатель

    return p;
}

void Push(struct Queue **list ,int element, int prio)
{
    struct Queue *tmp = *list; // Сохранение оригинального указателя на
голову

    if(tmp != NULL) // Проверка на существование списка
    {
        struct Queue *new_element = init(element, prio); // Создание нового
элемента

        while(tmp -> next != NULL)
        {
            tmp = tmp -> next;
        }

        tmp -> next = new_element;
    }
    else if(tmp == NULL) // Инициализация списка, если его нет
    {
        *list = init(element, prio);
    }
}

struct Queue *Priority(struct Queue *list)
{
    int n = 0, j = 0;

```

```

int num = 1;
struct Queue *tmp = list;

// Узнаём количество элементов в списке
while(tmp -> next != NULL)
{
    num++;
    tmp = tmp -> next;
}
tmp = list;

array = calloc(num, 4); // Создаём массив приоритетов

for(j = 0; j < num; j++)
{
    array[j] = tmp -> priority; // Запись приоритетов в массив
    tmp = tmp -> next;
}
tmp = list;

// Сортировка массива пузырьком (от большего к меньшему)
if(num > 1)
{
    for(j = 0; j < num - 1; j++)
    {
        for(int k = 0; k < num - 1; k++)
        {
            if(array[k] < array[k + 1])
            {
                n = array[k];
                array[k] = array[k + 1];
                array[k + 1] = n;
            }
        }
    }

    list = QueueSort(list, num); // Получение указателя на отсортированный
список

    return list;
}

struct Queue *QueueSort(struct Queue *list, int size)
{
    struct Queue *tmp = list, *pointer = NULL;
    int *arr = calloc(size, 4), i = 0; // Создаём массив для элементов

    // Проходим по массиву и списку, сравнивая приоритеты
    for(i = 0; i < size; i++)
    {
        tmp = list;
        while(tmp != NULL)
        {
            if(tmp -> priority == array[i])
            {
                Push(&pointer, tmp -> data, array[i]); // Запись элементов и
приоритетов в новую очередь
                tmp -> priority = 0; // Обнуление прочитанного приоритета
            }
            tmp = tmp -> next;
        }
    }

    // Освобождаем память
    free(arr);
    free(array);
}

```



```

        while(list != NULL)
        {
            struct Queue *to_delete = list;
            list = list -> next;
            free(to_delete);
        }

        return pointer;
    }

    struct Queue *Pop(struct Queue *list)
    {
        list = Priority(list);
        res = list -> data; // Получение элемента очереди
        pr = list -> priority; // Получение приоритета элемента очереди

        struct Queue *to_delete = list; // Переназначение указателя на первый
элемент
        list = list -> next; //Переназначение первого указателя на следующий
        free(to_delete); // Очистка памяти по предыдущему указателю

        return list; // Возвращение нового указателя на вершину очереди
    }

```

Файл Queue/main.c

```

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

struct Queue; // Структура, отвечающая за элементы очереди
struct Queue *init(int); // Инициализация очереди
void Push(struct Queue **, int); // Добавление элемента в очередь
struct Queue *Pop(struct Queue *); // Чтение элемента из очереди с
последующим удалением

int res = 0; // Результат извлечения из очереди

int main()
{
    int elem = 0, cont = 0;

    printf("\t# Queue #\n");
    printf("Enter element: ");
    scanf("%d", &elem);
    struct Queue *list = init(elem); // Инициализация очереди
    printf("Continue(any integer) or not(0): ");
    scanf("%d", &cont);
    if(cont == 0) goto printing;

    while(cont != 0)
    {
        printf("Enter element: ");
        scanf("%d", &elem);
        Push(&list, elem);
        printf("Continue(any integer) or not(0): ");
        scanf("%d", &cont);
    }
    printf("+-----+\n| elements |\n+-----+\n");
    while(list != NULL)
    {
        list = Pop(list);
        printf("|%9d |\n+-----+\n", res);
    }
    goto ret;

    printing:

```

```

        list = Pop(list);
        printf("+-----+\n|  element  |\n+-----+\n");
        printf("|%9d |\n+-----+\n", res);

        ret:
        return 0;
    }

    struct Queue
    {
        int data;    // Числовой элемент очереди
        struct Queue *next; // Указатель на следующий элемент очереди
    };

    struct Queue *init(int element)
    {
        struct Queue *p = NULL; // Создание указателя на структуру
        if((p = malloc(sizeof(struct Queue))) == NULL) // Выделение памяти под
структуру
        {
            printf("Unable to allocate memory: ");
            exit(1);
        }

        p -> data = element;    // Присваивание введённого значения полю данных
        p -> next = NULL;    // Установка на нулевой указатель

        return p;
    }

    void Push(struct Queue **list ,int element)
    {
        struct Queue *tmp = *list;    // Сохранение оригинального указателя на
голову

        if(tmp != NULL) // Проверка на существование списка
        {
            struct Queue *new_element = init(element); // Создание нового элемента

            while(tmp -> next != NULL)
            {
                tmp = tmp -> next;
            }

            tmp -> next = new_element;
        }
        else if(tmp == NULL)    // Инициализация списка, если его нет
        {
            *list = init(element);
        }
    }

    struct Queue *Pop(struct Queue *list)
    {
        res = list -> data; // Получение элемента очереди

        struct Queue *to_delete = list; // Переназначение указателя на первый
элемент
        list = list -> next;    //Переназначение первого указателя на следующий
        free(to_delete);    // Очистка памяти по предыдущему указателю

        return list;    // Возвращение нового указателя на вершину очереди
    }

```

Файл Stack/main.c

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

struct Stack; // Структура, отвечающая за элементы очереди
struct Stack *init(int); // Инициализация очереди
void Push(struct Stack **, int); // Добавление элемента в очередь
int Pop(struct Stack *); // Чтение элемента из очереди с последующим
удалением

int main()
{
    int result = 0, elem = 0, cont = 0, size = 0;

    printf("\t# Stack #\n");
    printf("Enter element: ");
    scanf("%d", &elem);
    struct Stack *list = init(elem); // Инициализация очереди
    printf("Continue(any integer) or not(0): ");
    scanf("%d", &cont);

    if(cont == 0) goto printing;

    while(cont != 0)
    {
        printf("Enter element: ");
        scanf("%d", &elem);
        Push(&list, elem);
        printf("Continue(any integer) or not(0): ");
        scanf("%d", &cont);
        size++;
    }

    printf("+-----+\n| elements |\n+-----+\n");
    for(int i = 0; i <= size; i++)
    {
        result = Pop(list);
        printf("|%9d |\n+-----+\n", result);
    }

    goto ret;

    printing:
    result = Pop(list);
    printf("+-----+\n| element |\n+-----+\n");
    printf("|%9d |\n+-----+\n", result);

    ret:
    return 0;
}

struct Stack
{
    int data; // Числовой элемент очереди
    struct Stack *next; // Указатель на следующий элемент очереди
};

struct Stack *init(int element)
{
    struct Stack *p = NULL; // Создание указателя на структуру

    if((p = malloc(sizeof(struct Stack))) == NULL) // Выделение памяти под
структуру
```

```

    {
        printf("Unable to allocate memory: ");
        exit(1);
    }

    p -> data = element;    // Присваивание введённого значения полю данных
    p -> next = NULL;    // Установка на нулевой указатель

    return p;
}

void Push(struct Stack **list ,int element)
{
    struct Stack *new_element = init(element); // Создание нового элемента

    struct Stack *tmp = *list; // Сохранение оригинального указателя на
голову

    while(tmp -> next != NULL)
    {
        tmp = tmp -> next;
    }

    tmp -> next = new_element;
}

int Pop(struct Stack *list)
{
    int res = 0;    // Результат извлечения из очереди
    int i = 0, j;    // Переменные счётчики числа элементов стека
    struct Stack *tmp = list, *p = list;    // Сохранение указателя на вершину
стека

    while(list -> next != NULL) // Обход списка
    {
        list = list -> next;
        i++;
    }
    i--;

    res = list -> data; // Получение элемента стека

    struct Stack *to_delete = list; // Переназначение указателя на последний
элемент

    for(j = 0; j < i; j++)
    {
        tmp = tmp -> next;    // Очередной обход списка до предпоследнего элемента
    }

    tmp -> next = NULL; // Присвоение предпоследнему элементу нулевого
указателя

    // Сохранение первого листа списка
    if(p -> next != NULL) free(to_delete);    // Очистка памяти по последнему
указателю

    return res;    // Возвращение значения
}

```

Приложение Б

Внесение данных в структуры

```
PriorityQueue

# Priority Queue #
Enter integer and priority(>0): 10 11
Continue(any integer) or not(0): 1
Enter integer and priority(>0): 5 3
Continue(any integer) or not(0): 1
Enter integer and priority(>0): 8 1
Continue(any integer) or not(0): 1
Enter integer and priority(>0): 6 3
Continue(any integer) or not(0): 1
Enter integer and priority(>0): 8 6
Continue(any integer) or not(0): 1
Enter integer and priority(>0): 100 2
Continue(any integer) or not(0): 0
```

Рисунок 1 — Внесение данных в приоритетную очередь

```
Lab3

# Queue #
Enter element: 10
Continue(any integer) or not(0): 1
Enter element: 101
Continue(any integer) or not(0): 1
Enter element: 5
Continue(any integer) or not(0): 1
Enter element: 6
Continue(any integer) or not(0): 1
Enter element: -70
Continue(any integer) or not(0): 0
```

Рисунок 2 — Внесение данных в очередь

```
Stack

# Stack #
Enter element: 10
Continue(any integer) or not(0): 1
Enter element: 5
Continue(any integer) or not(0): 1
Enter element: 7
Continue(any integer) or not(0): 1
Enter element: 8
Continue(any integer) or not(0): 1
Enter element: -13
Continue(any integer) or not(0): 1
Enter element: 1
Continue(any integer) or not(0): 0
```

Рисунок 3 — Внесение данных в стек