Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №3

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Динамические списки»

Выполнили:

студенты группы 22BВВ1

Митрошин Ю.Е

Коннов А.Д

Приняли:

Акифьев И.В

Юрова О.В

Пенза 2023

**Название**

Динамические списки

**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:

structnode

{

charinf[256];  // полезная информация

structnode \*next; // ссылка на следующий элемент

};

Обращение к списку и его элементам осуществляется посредством указателей:

structnode \*head = NULL, \*last = NULL, \*f = NULL; // указатели на первый и последний элементы списка

int dlinna = 0;

Для списка реализованы функции создания, добавления, удаления элемента, просмотра списка, нахождения нужного элемента списка:

// Функции добавления элемента, просмотра списка

void spstore(void), review(void), del(char \*name);

char find\_el[256];

struct node \*find(char \*name); // функциянахожденияэлемента

struct node \*get\_struct(void); // функциясозданияэлемента

struct node \*get\_struct(void)

{

struct node \*p = NULL;

char s[256];

if ((p = (node\*)malloc(sizeof(struct node))) == NULL)  // выделяемпамятьподновыйэлементсписка

{

printf("Ошибка при распределении памяти\n");

exit(1);

}

printf("Введите название объекта: \n");   // вводим данные

scanf("%s", s);

if (\*s == 0)

{

printf("Запись не была произведена\n");

return NULL;

}

strcpy(p->inf, s);

p->next = NULL;

return p; // возвращаем указатель на созданный элемент

}

/\* Последовательное добавление в список элемента (в конец)\*/

void spstore(void)

{

struct node \*p = NULL;

p = get\_struct();

if (head == NULL &&p != NULL) // если списка нет, то устанавливаем голову списка

{

head = p;

last = p;

}

else if (head != NULL && p != NULL) // список уже есть, то вставляем в конец

{

last->next = p;

last = p;

}

return;

}

/\* Просмотрсодержимогосписка. \*/

void review(void)

{

struct node \*struc = head;

if (head == NULL)

{

printf("Списокпуст\n");

}

while (struc)

{

printf("Имя - %s, \n", struc->inf);

struc = struc->next;

}

return;

}

/\* Поиск элемента по содержимому. \*/

struct node \*find(char \*name)

{

struct node \*struc = head;

if (head == NULL)

{

printf("Списокпуст\n");

}

while (struc)

{

if (strcmp(name, struc->inf) == 0)

{

return struc;

}

struc = struc->next;

}

printf("Элемент не найден\n");

return NULL;

}

/\* Удаление элемента по содержимому. \*/

voiddel(char \*name)

{

structnode \*struc = head; // указатель, проходящий по списку установлен на начало списка

structnode \*prev;// указатель на предшествующий удаляемому элемент

intflag = 0;      // индикатор отсутствия удаляемого элемента в списке

if (head == NULL) // если голова списка равна NULL, то список пуст

{

printf("Списокпуст\n");

return;

}

if (strcmp(name, struc->inf) == 0) // еслиудаляемыйэлемент - первый

{

flag = 1;

head = struc->next; // установливаем голову на следующий элемент

free(struc);  // удаляем первый элемент

struc = head; // устанавливаем указатель для продолжения поиска

}

else

{

prev = struc;

struc = struc->next;

}

while (struc) // проход по списку и поиск удаляемого элемента

{

if (strcmp(name, struc->inf) == 0) // еслинашли, то

{

flag = 1;         // выставляем индикатор

if (struc->next)  // если найденный элемент не последний в списке

{

prev->next = struc->next; // меняемуказатели

free(struc);     // удаляем  элемент

struc = prev->next; // устанавливаем указатель для продолжения поиска

}

else // если найденный элемент последний в списке

{

prev->next = NULL; // обнуляем указатель предшествующего элемента

free(struc); // удаляем  элемент

return;

}

}

else / если не нашли, то

{

prev = struc; // устанавливаем указатели для продолжения поиска

struc = struc->next;

}

}

if (flag == 0) // если флаг = 0, значит нужный элемент не найден

{

printf("Элемент не найден\n");

return;

}

}

**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).

**Листинг**

class PriorityQueueNode:

    def \_\_init\_\_(self, item, priority):

        self.item = item

        self.priority = priority

        self.next = None

class PriorityQueue:

    def \_\_init\_\_(self):

        self.head = None

    def is\_empty(self):

        return self.head is None

    def enqueue(self, item, priority):

        new\_node = PriorityQueueNode(item, priority)

        if self.head is None or priority >self.head.priority:

            new\_node.next = self.head

            self.head = new\_node

        else:

            current = self.head

            while current.next is not None and priority <= current.next.priority:

                current = current.next

            new\_node.next = current.next

            current.next = new\_node

    def dequeue(self):

        if not self.is\_empty():

            item = self.head.item

            self.head = self.head.next

            return item

pq = PriorityQueue()

while(True):

    task = input("введите задачу: ")

    if task == ".":

        break

    else:

        priority = int(input("введите приоритет задачи: "))

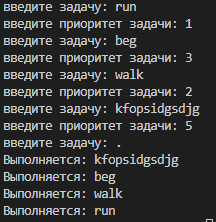
        pq.enqueue(task, priority)

while not pq.is\_empty():

    item = pq.dequeue()

    print("Выполняется:", item)

Результат работы программы:



**Вывод:** в ходе лабораторной работы мы изучили динамические списки и научились работать с ними.