МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет електроніки та комп’ютерних технологій

Про виконання лабораторної роботи №4
Поняття про динамічну пам’ять та вказівники.

Виконав:
студент групи
ФеП-13
Жгута Денис

Львів-2023

Мета: повторити вивчений матеріал щодо поняття динамічної пам'яті та вказівників, вивчити що таке динамічні структури даних, лінійні списки та їхні загальні положення, стеки лінійних списків.

Теоретичні відомості: до динамічних структур даних належать такі структури, розміри яких визначаються і можуть змінюватися протягом виконання програми. Основними видами динамічних структур є лінійні списки, нелінійні списки (дерева) та графи.

Зв’язний лінійний список – це набір однотипних компонентів, які зв’язані між собою за допомогою вказівників. Зв’язані лінійні списки бувають таких типів:

- однозв’язний лінійний список;
- двозв’язний лінійний список;
- стек;
- черга:
- дек.

Однозв’язний лінійний список – це список, в якому попередній компонент посилається на наступний. У випадку. коли в однозв’язному лінійному списку останній елемент посилається на перший, утворюється однозв’язний циклічний список. Двозв’язний лінійний список – це список, в якому попередній компонент посилається на наступний, а наступний – на попередній. У випадку. коли в двозв’язному лінійному списку останній елемент посилається на перший, а перший – на останній, утворюється двозв’язний циклічний список. Стек – це однозв’язний лінійний список, в якому компоненти додаються та видаляються лише з його вершини, тобто з початку списку. Черга – це однозв’язний лінійний список, в якому компоненти додаються в кінець списку, а видаляються з вершини, тобто з початку списку.
Дек (англ. Double ended queue – двобічна черга) – це черга, в якій елементи можуть додаватися і видалятися з обох кінців. Стек (англ. stack) – це однозв’язний лінійний список, в якому доступ (вставка/видалення) до його компонент здійснюється через початок (вершину – head) списку. Стек працює за принципом LIFO (від англ. Last In First Out – останній прийшов – перший пішов).

ХІД РОБОТИ

Для виконання лабораторної роботи використовую мову програмування “Python”, для цього створюю два файли main.py і List.py. Реалізацію наступних алгоритмів демонструю кодом нижче. Для реалізації останнього завдання створю окремі файли main2.py та List2.py (для зручності реалізації).

main.py

```python
from List import Stack

from List import Queue



def main():

   queue = Queue()

   stack = Stack()

   while True:

       print("1. Додати елемент в стек")

       print("2. Видалити елемент зі стеку")

       print("3. Показати вміст стеку")

       print("4. Додати елемент в чергу")

       print("5. Видалити елемент з черги")

       print("6. Показати вміст черги")

       print("7. Вийти з програми")

       choice = int(input("Виберіть дію: "))

       if choice == 1:

           data = input("Введіть дані для додавання: ")

           stack.push(data)

           stack.show()

       elif choice == 2:

           popped_element = stack.pop()

           if popped_element is None:

               print("Стек порожній")

           else:

               print("Видалено: ", popped_element)
```

```
                stack.show()

        elif choice == 3:

            stack.show()

        elif choice == 4:

            data = input("Введіть дані для додавання: ")

            queue.enqueue(data)

            queue.display()

        elif choice == 5:

            popped_element = queue.dequeue()

            if popped_element is None:

                print("Черга порожня")

            else:

                print("Видалено: ", popped_element)

                queue.display()

        elif choice == 6:

            queue.display()

        elif choice == 7:

            break

        else:

            print("Невірний вибір")




if __name__ == '__main__':

    main()
```

List.py

```
class Node:

   def __init__(self, data):

       self.data = data

       self.next = None



# class Queue:

#     def __init__(self):

#         self.data = []



#     def is_empty(self):

#         return len(self.data) == 0



#     def enqueue(self, data):

#         self.data.append(data)



#     def dequeue(self):

#         if self.is_empty():

#             return None

#         return self.data.pop(0)



#     def display(self):

#         print(self.data)
```

```
class Queue:

    def __init__(self):

        self.head = None

        self.tail = None



    def is_empty(self):

        return self.head is None



    def enqueue(self, data):

        new_node = Node(data)

        if self.is_empty():

            self.head = new_node

            self.tail = new_node

        else:

            self.tail.next = new_node

            self.tail = new_node



    def dequeue(self):

        if self.is_empty():

            return None

        popped_node = self.head

        self.head = self.head.next

        if self.head is None:

            self.tail = None

        popped_node.next = None

        return popped_node.data
```

```
    def display(self):
        current = self.head
        while current:
            print(current.data, end=' ')
            current = current.next
        print()



class Stack:
    def __init__(self):
        self.head = None


    def push(self, data):
        new_node = Node(data)
        new_node.next = self.head
        self.head = new_node


    def pop(self):
        if self.head is None:
            return None
        popped_node = self.head
        self.head = self.head.next
        popped_node.next = None
        return popped_node.data
```

```
   def show(self):
        current = self.head
        while current:
            print(current.data, end=' ')
            current = current.next
        print()


#v2
# class Stack:
#     def __init__(self):
#         self.stack = []

#     def push(self, data):
#         self.stack.insert(0, data)

#     def pop(self):
#         if not self.is_empty():
#             return self.stack.pop()

#     def is_empty(self):
#         return len(self.stack) == 0

#     def show(self):
#         print(self.stack)
```

Також демонструю невеличкі результати роботи моєї програми:

```
5. Видалити елемент з черги
6. Показати вміст черги
7. Вийти з програми
Виберіть дію: 4
Введіть дані для додавання: 1
1
1. Додати елемент в стек
2. Видалити елемент зі стеку
3. Показати вміст стеку
4. Додати елемент в чергу
5. Видалити елемент з черги
6. Показати вміст черги
7. Вийти з програми
Виберіть дію: 4
Введіть дані для додавання: 10
1 10
1. Додати елемент в стек
2. Видалити елемент зі стеку
3. Показати вміст стеку
4. Додати елемент в чергу
5. Видалити елемент з черги
6. Показати вміст черги
7. Вийти з програми
Виберіть дію: 5
Видалено:  1
10
1. Додати елемент в стек
2. Видалити елемент зі стеку
3. Показати вміст стеку
4. Додати елемент в чергу
5. Видалити елемент з черги
6. Показати вміст черги
7. Вийти з програми
Виберіть дію:
```

```
5. Видалити елемент з черги
6. Показати вміст черги
7. Вийти з програми
Виберіть дію: 1
Введіть дані для додавання: 5
5
1. Додати елемент в стек
2. Видалити елемент зі стеку
3. Показати вміст стеку
4. Додати елемент в чергу
5. Видалити елемент з черги
6. Показати вміст черги
7. Вийти з програми
Виберіть дію: 1
Введіть дані для додавання: 10
10 5
1. Додати елемент в стек
2. Видалити елемент зі стеку
3. Показати вміст стеку
4. Додати елемент в чергу
5. Видалити елемент з черги
6. Показати вміст черги
7. Вийти з програми
Виберіть дію: 2
Видалено:  10
5
1. Додати елемент в стек
2. Видалити елемент зі стеку
3. Показати вміст стеку
4. Додати елемент в чергу
5. Видалити елемент з черги
6. Показати вміст черги
7. Вийти з програми
Виберіть дію:
```

Тепер демонструю роботу наступної програми, код нижче:

main2.py

```
from List2 import LinkedList

def main():
   my_list = LinkedList()
   while True:
       print("\nDoubly Linked List Menu")
       print("1. Add element to beginning of list")
       print("2. Add element to end of list")
       print("3. Delete element from beginning of list")
       print("4. Delete element from end of list")
       print("5. Search for element in list")
       print("6. Add element to list after a specified element")
       print("7. Delete a specified element from list")
       print("8. Exit program")
       choice = int(input("Enter your choice: "))

       if choice == 1:
           data = input("Enter data for the new element: ")
           my_list.add_begin(data)
           my_list.display()
       elif choice == 2:
           data = input("Enter data for the new element: ")
           my_list.add_end(data)
           my_list.display()
       elif choice == 3:
           my_list.del_begin()
           my_list.display()
       elif choice == 4:
           my_list.del_end()
           my_list.display()
       elif choice == 5:
           key = input("Enter the key to search for: ")
           node = my_list.search(key)
           if node is None:
               print("Element not found")
           else:
               print("Element found")
       elif choice == 6:
           key = input("Enter the key after which to insert the new element: ")
           data = input("Enter data for the new element: ")
           my_list.add_mid(key, data)
           my_list.display()
       elif choice == 7:
```

```
            key = input("Enter the key of the element to delete: ")
            my_list.del_mid(key)
            my_list.display()
        elif choice == 8:
            print("Exiting program...")
            break
        else:
            print("Invalid choice. Try again.")

if __name__ == '__main__':
    main()
```

List2.py

```
class Node:
    def __init__(self, data):
        self.data = data
        self.prev = None
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None
        self.tail = None

    def is_empty(self):
        return self.head is None

    def add_begin(self, data):
        new_node = Node(data)
        if self.is_empty():
            self.head = new_node
            self.tail = new_node
        else:
            new_node.next = self.head
            self.head.prev = new_node
            self.head = new_node

    def add_end(self, data):
        new_node = Node(data)
        if self.is_empty():
            self.head = new_node
            self.tail = new_node
        else:
            self.tail.next = new_node
```

```
            new_node.prev = self.tail
            self.tail = new_node

    def del_begin(self):
        if self.is_empty():
            return
        if self.head == self.tail:
            self.head = None
            self.tail = None
        else:
            self.head = self.head.next
            self.head.prev = None

    def del_end(self):
        if self.is_empty():
            return
        if self.head == self.tail:
            self.head = None
            self.tail = None
        else:
            self.tail = self.tail.prev
            self.tail.next = None

    def search(self, key):
        curr_node = self.head
        while curr_node is not None:
            if curr_node.data == key:
                return curr_node
            curr_node = curr_node.next
        return None

    def add_mid(self, key, data):
        curr_node = self.search(key)
        if curr_node is None:
            return
        new_node = Node(data)
        new_node.next = curr_node.next
        new_node.prev = curr_node
        if curr_node.next is not None:
            curr_node.next.prev = new_node
        curr_node.next = new_node

    def del_mid(self, key):
        curr_node = self.search(key)
        if curr_node is None:
```

```
            return
        if curr_node == self.head:
            self.del_begin()
        elif curr_node == self.tail:
            self.del_end()
        else:
            curr_node.prev.next = curr_node.next
            curr_node.next.prev = curr_node.prev

    def display(self):
        curr_node = self.head
        while curr_node is not None:
            print(curr_node.data, end=' ')
            curr_node = curr_node.next
        print()
```

Невеликі результати роботи програми:

```
Doubly Linked List Menu
1. Add element to beginning of list
2. Add element to end of list
3. Delete element from beginning of list
4. Delete element from end of list
5. Search for element in list
6. Add element to list after a specified element
7. Delete a specified element from list
8. Exit program
Enter your choice: 1
Enter data for the new element: 5
5 10 10

Doubly Linked List Menu
1. Add element to beginning of list
2. Add element to end of list
3. Delete element from beginning of list
4. Delete element from end of list
5. Search for element in list
6. Add element to list after a specified element
7. Delete a specified element from list
8. Exit program
Enter your choice: 4
5 10

Doubly Linked List Menu
1. Add element to beginning of list
2. Add element to end of list
3. Delete element from beginning of list
4. Delete element from end of list
5. Search for element in list
6. Add element to list after a specified element
7. Delete a specified element from list
8. Exit program
Enter your choice: 5
Enter the key to search for: 5
Element found

Doubly Linked List Menu
1. Add element to beginning of list
2. Add element to end of list
3. Delete element from beginning of list
4. Delete element from end of list
5. Search for element in list
6. Add element to list after a specified element
7. Delete a specified element from list
8. Exit program
Enter your choice: 1
Enter data for the new element: 1
1 5 10

Doubly Linked List Menu
1. Add element to beginning of list
2. Add element to end of list
3. Delete element from beginning of list
4. Delete element from end of list
5. Search for element in list
6. Add element to list after a specified element
7. Delete a specified element from list
8. Exit program
Enter your choice: 6
Enter the key after which to insert the new elemer
Enter data for the new element: 24
1 5 24 10

Doubly Linked List Menu
1. Add element to beginning of list
2. Add element to end of list
3. Delete element from beginning of list
4. Delete element from end of list
5. Search for element in list
6. Add element to list after a specified element
7. Delete a specified element from list
8. Exit program
Enter your choice: 7
Enter the key of the element to delete: 10
1 5 24

Doubly Linked List Menu
1. Add element to beginning of list
2. Add element to end of list
3. Delete element from beginning of list
4. Delete element from end of list
5. Search for element in list
6. Add element to list after a specified element
7. Delete a specified element from list
8. Exit program
Enter your choice:
```

Висновок: на цій лабораторній роботі я зрозумів що таке стек, черга та двозв'язний список, а також різницю між ними. Також під час виконання лабораторної роботи я навчився використовувати структури та створювати Ноди в Пайтоні, зв’язувати їх, а також працювати з елементами списку. Незважаючи на те, що в мові програмування Пайтон вже давно є готові функції, котрими можна легко це все реалізувати, я все ж вирішив самотужки написати код цих функцій. Мабуть це навіть і краще.