Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Российский химико-технологический университет имени Д.И. Менделеева»
Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3

Выполнил студент группы.....КС-33..... (Вагенлейтнер Никита Сергеевич)

Ссылка на репозиторий: .....(https://github.com/MUCTR-IKT-CPP/VagenlejtnerNS_33_alg.git)

Приняли: ..... Пысин Максим Дмитриевич
.....Краснов Дмитрий Олегович
.....Лобанов Алексей Владимирович
.....Крашенинников Роман Сергеевич

Дата сдачи: ..... (26.02.2025)

Оглавление

Описание задачи.

Написать две реализации стека (Стек через односвязный список и стек через массив)

- Добавление элемента в начало
- Взятие элемента из начала

В рамках лабораторной работы необходимо изучить и реализовать одну из трёх структур(двухсвязный список, стек, очередь), в соответствии со своим вариантом, при этом, все структуры должны:

Использовать шаблонный подход, обеспечивая работу контейнера с произвольными данными.

Реализовывать свой итератор предоставляющий стандартный для языка механизм работы с ним(для C++ это операции ++ и операция !=, для python это)

Обеспечивать работу стандартных библиотек и конструкции for each если она есть в языке, если их нет, то реализовать собственную функцию использующую итератор.

Проверку на пустоту и подсчет количества элементов.

Для демонстрации работы структуры необходимо создать набор тестов(под тестом понимается функция, которая создаёт структуру, проводит операцию или операции над структурой и удаляет структуру):

заполнение контейнера 1000 целыми числами в диапазоне от -1000 до 1000 и подсчет их суммы, среднего, минимального и максимального.

Провести проверку работы операций вставки и изъятия элементов на коллекции из 10 строковых элементов.

заполнение контейнера 100 структур содержащих фамилию, имя, отчество и дату рождения(от 01.01.1980 до 01.01.2020) значения каждого поля генерируются случайно из набора заранее заданных. После заполнение необходимо найти всех людей младше 20 лет и старше 30 и создать новые структуры содержащие результат фильтрации, проверить выполнение на правильность подсчётом кол-ва элементов не подходящих под условие в новых структурах. Тесты по вариантам:

- Инверсировать содержимое контейнера заполненного отсортированными по возрастанию элементами не используя операцию перемещения при помощи итератора, а только операторы изъятия и вставки.
- Сравнить две реализации между собой (Сравнить на основании скорости выполнения операции вставки и изъятия на контейнере, использования памяти на все элементы), тестировать для коллекции состоящей из 10000 элементов.

Описание метода/модели.

Стек

Стек через односвязный список

Стек реализуется с помощью узлов односвязного списка, где каждый узел содержит значение и ссылку на следующий элемент. Операции push и pop выполняются за O(1), так как добавление и удаление происходит на вершине стека. Использование динамической памяти позволяет стеку расти без ограничения, но требует дополнительных затрат на хранение ссылок.

Стек через массив

Стек реализуется в виде массива фиксированного или динамического размера. Операции push и pop также выполняются за O(1), но могут потребовать реаллокации при переполнении (если используется динамический массив). Преимуществом является компактность хранения данных, но стек ограничен размером массива.

Выполнение задачи.

Для реализации и выполнения данного алгоритма был задействован язык Python. Были реализованы следующие программные блоки: алгоритмический блок, тестовый блок, аналитический блок полученных результатов работы алгоритма.

Блок кода отвечающий за реализацию стека:

```
# ---------- Реализация стека через массив ----------
class ArrayStack(Generic[T]):
    def __init__(self):
        self._data: List[T] = []
    def push(self, value: T) -> None:
        self._data.append(value)
    def pop(self) -> T:
        if not self._data:
            raise IndexError("pop from empty stack")
        return self._data.pop()
    def is_empty(self) -> bool:
        return len(self._data) == 0
    def __len__(self) -> int:
        return len(self._data)
    def __iter__(self) -> Iterator[T]:
        return reversed(self._data)
# ---------- Реализация стека через односвязный список ----------
class LinkedListStack(Generic[T]):
    class _Node(Generic[T]):
        def __init__(self, value: T, next_: Optional['LinkedListStack._Node'] = None):
            self.value = value
            self.next = next_
    def __init__(self):
        self._head: Optional[LinkedListStack._Node[T]] = None
        self._size = 0
    def push(self, value: T) -> None:
        new_node = self._Node(value, self._head)
        self._head = new_node
        self._size += 1
    def pop(self) -> T:
        if self._head is None:
            raise IndexError("pop from empty stack")
        value = self._head.value
        self._head = self._head.next
        self._size -= 1
        return value
    def is_empty(self) -> bool:
        return self._head is None
    def __len__(self) -> int:
        return self._size
```

Блоки кода отвечающие за тестирование:

```
# Тест 1: Заполнение контейнера числами и подсчет характеристик
def test_numeric_operations(stack_class: Any):
    stack = stack_class[int]()
    # Заполнение контейнера
    for _ in range(1000):
        stack.push(random.randint(-1000, 1000))
    # Итерация и подсчеты
    total = 0
    count = 0
    minimum = float('inf')
    maximum = float('-inf')
    for value in stack:
        total += value
        count += 1
        minimum = min(minimum, value)
        maximum = max(maximum, value)
    average = total / count if count > 0 else 0
    print(f"Test Numeric Operations ({stack_class.__name__}):")
    print(f"  Total: {total}, Average: {average}, Min: {minimum}, Max: {maximum}")
    return total, average, minimum, maximum
# Тест 2: Проверка строковых данных
def test_string_operations(stack_class: Any):
    stack = stack_class[str]()
    strings = ["one", "two", "three", "four", "five", "six", "seven", "eight", "nine", "ten"]
    # Вставка элементов
    for s in strings:
        stack.push(s)
    # Извлечение и проверка
    results = []
    while not stack.is_empty():
        results.append(stack.pop())
    print(f"Test String Operations ({stack_class.__name__}):")
    print(f"  Results: {results}")
# Структура для тестирования данных о людях
Person = namedtuple('Person', ['last_name', 'first_name', 'patronymic', 'birth_date'])
# Генерация случайного человека
def random_person():
    last_names = ["Ivanov", "Petrov", "Sidorov", "Smirnov"]
    first_names = ["Ivan", "Petr", "Alexey", "Dmitry"]
    patronymics = ["Ivanovich", "Petrovich", "Alexeevich", "Dmitrievich"]
```

Результаты работы программы выводится в output строке и приведены ниже.

```
  Test Numeric Operations (ArrayStack):
    Total: 20172, Average: 20.172, Min: -997, Max: 999
  Test Numeric Operations (LinkedListStack):
    Total: 13832, Average: 13.832, Min: -996, Max: 998
  Test String Operations (ArrayStack):
    Results: ['ten', 'nine', 'eight', 'seven', 'six', 'five', 'four', 'three',
'two', 'one']
  Test String Operations (LinkedListStack):
    Results: ['ten', 'nine', 'eight', 'seven', 'six', 'five', 'four', 'three',
'two', 'one']
  Test Person Data (ArrayStack):
    Below 20: 32, Above 30: 39
  Test Person Data (LinkedListStack):
    Below 20: 27, Above 30: 47
  Test Inversion (ArrayStack):
    Original: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
    Inverted: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
  Test Inversion (LinkedListStack):
    Original: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
    Inverted: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
  Comparison:
    ArrayStack Time: 0.00109 s
    LinkedListStack Time: 0.00285 s
```

Заключение.

Стек через односвязный список удобен при работе с неизвестным количеством данных, так как не требует предварительного выделения памяти и может динамически расти. Однако он требует дополнительных затрат на хранение ссылок.

Стек через массив более эффективен по памяти и работает быстрее за счёт локальности данных, но его размер либо фиксирован, либо требует затрат на реаллокацию при расширении. Выбор реализации зависит от требований к памяти и скорости работы.