Лабораторная работа № 6 - АТД. Контейнеры

Цель работы

1. Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
2. Реализация класса-контейнера.

Постановка задачи

1. Определить класс-контейнер.
2. Реализовать конструкторы, деструктор, операции ввода-вывода, операцию присваивания.
3. Перегрузить операции, указанные в варианте.
4. Реализовать класс-итератор. Реализовать с его помощью операции последовательного доступа.
5. Написать тестирующую программу, иллюстрирующую выполнение операций.

> Класс- контейнер МНОЖЕСТВО с элементами типа int. Реализовать операции:
>
> [] – доступа по индексу;
>
> != - проверка на неравенство;
>
> < число – принадлежность числа множеству;
>
> +n – переход вправо к элементу с номером n (с помощью класса-итератора).

Описание класса-контейнера

```
// Множество
class Array {
public:
explicit Array(int s, int k = 0);

Array(const Array &a);

~Array();

Array &operator=(const Array &a);     //оператор присваивания
int &operator[](int index); //операция доступа по индексу
Array operator+(const int k);
int operator()(); //операция, возвращающая длину множества
friend ostream &operator<<(ostream &out, const Array &a); //перегруженные операции
ввода-вывода
friend istream &operator>>(istream &in, Array &a);

Iterator first() { return beg; }//возвращает указатель на первый элемент
Iterator last() {
return end; }//возвращает указатель на элемент следующий за
private:
```

```
int size;//размер множества
int *data;//укзатель на динамический массив значений множества
Iterator beg;//указатель на первый элемент множества
Iterator end;//указатель на элемент следующий за последним
};
```

Определение компонентных функций

```
Array::Array(int s, int k) {
    size = s;
    data = new int[size];
    for (int i = 0; i < size; i++)
        data[i] = k;
    beg.elem = &data[0];
    end.elem = &data[size];
}

//конструктор копирования
Array::Array(const Array &a) {
    size = a.size;
    data = new int[size];
    for (int i = 0; i < size; i++)
        data[i] = a.data[i];
    beg = a.beg;
    end = a.end;
}

//деструктор
Array::~Array() {
    delete[]data;
    data = 0;
}

//операция присваивания
Array &Array::operator=(const Array &a) {
    if (this == &a)return *this;
    size = a.size;
    if (data != 0) delete[]data;
    data = new int[size];
    for (int i = 0; i < size; i++)
        data[i] = a.data[i];
    beg = a.beg;
    end = a.end;
    return *this;
}

//операция доступа по индексу
int &Array::operator[](int index) {
    if (index < size) return data[index];
    else cout << "\nError! Index>size";
}
```

```
//операция для получения длины множества
int Array::operator()() {
    return size;
}

//операции для ввода-выода
ostream &operator<<(ostream &out, const Array &a) {
    for (int i = 0; i < a.size; ++i)
        out << a.data[i] << " ";
    return out;
}

istream &operator>>(istream &in, Array &a) {
    for (int i = 0; i < a.size; ++i)
        in >> a.data[i];
    return in;
}

//операция для добавления константы
Array Array::operator+(const int k)//+k
{
    Array temp(size);
    for (int i=0;i<size;++i)
        temp.data[i]+=data[i]+k;
    return temp;
}
```

Описание класса-итератора и его компонентных функций

```
class Iterator {
    friend class Array;                    //дружественный класс
public:
    Iterator() { elem = nullptr; }               //конструктор без параметров
    Iterator(const Iterator &it) { elem = it.elem; }//конструктор копирования
    //перегруженные операции сравнения
    bool operator==(const Iterator &it) { return elem == it.elem; }
    bool operator!=(const Iterator &it) { return elem != it.elem; };

    void operator++(int) { ++elem; };            //перегруженная операция
инкремент
    void operator--(int) { --elem; }              //перегруженная операция
декремент
    int &operator*() const { return *elem; }  //перегруженная операция
разыменования
private:
    int *elem;                                 //указатель на элемент типа int
};
```

Функция main()

```
int main() {
    Array a(5);          //создали множество из 5 элементов, заполненный нулями
    cout << a << "\n";      //вывели значения элементов множества
    cin >> a;             //ввели с клавиатуры значения элементов множества
    cout << a << "\n";      //вывели значения элементов множества
    a[2] = 100;           //используя операцию [] присвоили новое значение
элементу
    cout << a << "\n";      //вывели значения элементов множества
    Array b(10);         //создали множество b из 10 элементов, заполненный нулями
    cout << b << "\n";      //вывели значения элементов множества
    b = a;                //присвоили множеству b значения множества a
    cout << b << "\n";      //вывели значения элементов множества
    Array c(10);         //создали множество c из 10 элементов, заполненный нулями
    c = b + 100;            //Увеличили значения множества b на 100 и присвоили
множеству c
    cout << c << "\n";      //вывели значения элементов множества c
    cout << "\nthe length of a=" << a() << endl;//вывели длину множества a
    //разыменовываем значение, которое возвращает a.first() и выводим его
    cout << *(a.first()) << endl;
    //переменную типа Iterator устанавливаем на первый элемент множества a с
    //помощью метода first
    Iterator i = a.first();
    //оперция инкремент
    i++;
    //разыменовываеи итератор и выводи его значение
    cout << *i << endl;
    //выводим значения элеменов множества с помощью итератора
    for (i = a.first(); i != a.last(); i++)cout << *i << endl;
    return 0;
}
```

Объяснение результатов работы программы

```
<< 0 0 0 0 0
>> 0
>> 0
>> 0
>> 0
>> 0
<< 0 0 0 0 0
<< 0 0 100 0 0
<< 0 0 0 0 0 0 0 0 0 0
<< 0 0 100 0 0
<< 100 100 200 100 100
<<
<< the length of a=5
<< 0
<< 0
<< 0
<< 0
```

```
<< 100
<< 0
<< 0

Process finished with exit code 0
```

1. Что такое абстрактный тип данных? Привести примеры АТД.

Ответ: АТД - тип данных, определяемый только через операции, которые могут выполняться над соответствующими объектами безотносительно к способу представления этих объектов.

Примеры:

```
- Стек (операции push, pop, peek)
- Очередь (enqueue, dequeue)
- Множество (add, remove, contains)
- Список (insert, delete, get)
```

2. Привести примеры абстракции через параметризацию.

Ответ: Абстракция через параметризацию может быть осуществлена так же, как и для процедур (функций); использованием параметров там, где это имеет смысл.

Прмиеры:

- Шаблоны классов в C++ (template)
- Функции с параметрами, которые могут принимать разные типы данных
- Контейнеры STL (vector, list, set), которые могут работать с разными типами элементов

3. Привести примеры абстракции через спецификацию.

Ответ: Абстракция через спецификацию достигается за счет того, что операции представляются как часть типа.

Примеры:

- Интерфейсы (чисто виртуальные классы в C++)
- Абстрактные классы с виртуальными методами
- Заголовочные файлы (.h), которые определяют сигнатуры функций без их реализации

4. Что такое контейнер? Привести примеры.

Ответ: Контейнер – набор однотипных элементов. Встроенные массивы в C++ - частный случай контейнера.

Примеры:

- Встроенные массивы C++
- Вектор (std::vector)

- Список (std::list)
- Множество (std::set)
- Ассоциативные массивы (std::map)

5. Какие группы операций выделяют в контейнерах?

Ответ: Среди всех операций контейнера можно выделить несколько типовых групп:

1. Операции доступа: operator[], at(), front(), back()
2. Операции модификации: insert(), erase(), push_back(), pop_back()
3. Операции емкости: size(), empty(), capacity()
4. Операции итерации: begin(), end(), rbegin(), rend()
5. Операции сравнения: ==, !=, <, > и другие

6. Какие виды доступа к элементам контейнера существуют? Привести примеры.

Ответ: Основные виды доступа:

1. Прямой доступ по индексу (массивы, vector) - arr[3]
2. Последовательный доступ (списки) - с помощью итераторов
3. Ассоциативный доступ (map, set) - по ключу map["key"]
4. Доступ через итераторы - for(auto it = v.begin(); it != v.end(); ++it)

7. Что такое итератор?

Ответ: Итератор - это объект, который предоставляет доступ к элементам контейнера и позволяет перебирать их. Он абстрагирует способ доступа к элементам, предоставляя единый интерфейс для разных типов контейнеров.

8. Каким образом может быть реализован итератор?

Ответ: Итератор может быть реализован:

1. Как указатель (для массивов)
2. Как отдельный класс с перегруженными операциями (++, --, *, ->, ==, !=)
3. Как вложенный класс контейнера
4. С использованием стандартных итераторов STL (input, output, forward, bidirectional, random access)

9. Каким образом можно организовать объединение контейнеров?

Ответ: Объединение контейнеров можно организовать:

1. Методами контейнеров (insert, merge)
2. Алгоритмами STL (merge, set_union)
3. Перегрузкой операторов (operator+, operator+=)
4. Созданием нового контейнера и копированием элементов

10. Какой доступ к элементам предоставляет контейнер, состоящий из элементов «ключ-значение»?

Ответ: Контейнеры "ключ-значение" (map, unordered_map) предоставляют:

1. Доступ по ключу (operator[], at())
2. Поиск по ключу (find())
3. Доступ через итераторы к парам ключ-значение
4. Не предоставляют прямой доступ по индексу (кроме случаев, когда ключ - это индекс)

11. Как называется контейнер, в котором вставка и удаление элементов выполняется на одном конце контейнера?

Ответ: Такой контейнер называется стек (LIFO - Last In First Out). Пример - std::stack.

12. Какой из объектов (a,b,c,d) является контейнером?

```
d. 4. int mas[100];
```

Ответ:

13. Какой из объектов (a,b,c,d) не является контейнером?

```
d. 4. int mas;
```

Ответ:

14. Контейнер реализован как динамический массив, в нем определена операция доступ по индексу. Каким будет доступ к элементам контейнера?

Ответ: Прямой доступ (random access) с постоянной сложностью O(1), как в std::vector.

15. Контейнер реализован как линейный список. Каким будет доступ к элементам контейнера?

Ответ: Последовательный доступ (sequential access) с линейной сложностью O(n), как в std::list. Для доступа к n-ному элементу нужно пройти все предыдущие.

Лабораторная работа доступна в GitHub репозитории hanriel/PSTU-CPP