Лабораторная работа №6

АТД. Контейнеры

1. Цель задания:

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

2. Теоретические сведения

2.1. Абстрактные типы данных. Контейнеры.

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

АТД включает в себя абстракцию как через параметризацию, так и через спецификацию.

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

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

Для реализации АТД необходимо, во-первых, выбрать представление памяти для объектов и, во-вторых, реализовать операции в терминах выбранного представления.

Примером абстрактного типа данных является класс в языке C++.

В реальных задачах требуется обрабатывать группы данных большого объема, поэтому в любом языке программирования существуют средства, позволяющие объединять данные в группы. В первую очередь это массивы. В C++ массивы – это очень простые конструкции. В ООП этого недостаточно для работы с группами однородных данных. Поэтому была выработана более общая концепция объединения однородных данных в группу – контейнер.

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

Контейнер – это объект. Имя контейнера – это имя переменной. Контейнер, так же как и другие объекты, обладает временем жизни. Время жизни контейнера в общем случае не зависит от времени жизни его элементов. Элементами контейнера могут любые объекты, в том числе, и другие контейнеры.

Контейнеры могут быть фиксированного и переменного размера. В контейнере фиксированного размера число элементов постоянное, оно обычно задается при создании контейнера. Примером такого контейнера является массив. Для контейнера переменного размера количество элементов при объявлении обычно не задается. Элементы в таком контейнере добавляются и удаляются во время работы программы. Примером является список.

Если элементы контейнера не упорядочены, то добавление и удаление элементов обычно выполняется в начале и в конце контейнера. Способ вставки и удаления определяет вид контейнера. Если вставка и удаление осуществляется на одном конце, то такой контейнер называется стеком (Last In First Out – последним пришел, первым ушел). Если элементы добавляются на одном конце контейнера, а удаляются на другом, то такой контейнер называется очередью (First In First Out – первым пришел, последним ушел). Можно выполнять вставку и удаление на обоих концах контейнера, тогда такой контейнер будет называться двусторонней очередью (deque – double ended queue).

Если контейнер каким то образом упорядочен, то операция вставки работает в соответствии с порядком элементов в контейнере. Операция удаления может выполняться по разному: в начале, в конце или удаление заданного элемента.

2.2. Операции контейнера

Среди всех операций контейнера можно выделить несколько типовых групп:

- Операции доступа к элементам, которые обеспечивают и операцию замены значений элементов;
- Операции добавления и удаления элементов или групп элементов;
- Операции поиска элементов и групп элементов;
- Операции объединения контейнеров;
- Специальные операции, которые зависят от вида контейнера.

2.2.1. Доступ к элементам.

Доступ к элементам контейнера бывает: последовательный, прямой и ассоциативный.

Прямой доступ – это доступ по индексу. Например, a[10] – требуется найти элемент контейнера с номером 10. В C++ нумерацию элементов контейнера принято начинать с нуля.

Ассоциативный доступ также выполняется по индексу, но индексом будет являться не номер элемента, а его содержимое. Пусть имеется контейнер –словарь, в котором хранится информация, состоящая, как минимум из двух полей: слово и его перевод. Индексом может служить слово, например, a[“word”]. С этим словом будет связано слово-перевод. Поле, с содержимым которого ассоциируется элемент контейнера, называется ключом или полем доступа. Элемент, с которым ассоциируется ключ, называется значением. Контейнер, который представляет ассоциативный доступ, состоит из пар «ключ-значение». Ассоциативный контейнер каким-то образом должен быть упорядочен по ключу. Например, в словаре упорядочение выполняется по алфавиту.

При последовательном доступе осуществляется перемещение от элемента к элементу контейнера. Набор операций последовательного доступа включает следующие:

- Перейти к первому элементу;
- Перейти к последнему элементу;
- Перейти к следующему элементу;
- Перейти к предыдущему элементу;
- Перейти на n элементов вперед;
- Перейти на n элементов назад;
- Получить текущий элемент.

Если контейнером является массив, то мы можем осуществлять последовательный доступ к его элементам с помощью указателя. И все перечисленные операции можно реализовать, используя указатели. В более общем случае, объект, который перебирает элементы контейнера, называется итератором. Итератор – это объект, который обеспечивает последовательный доступ к элементам контейнера. Итератор может быть реализован как часть класса-контейнера в виде набора методов:

v.first()	перейти к первому элементу
v.last()	перейти к последнему элементу
v.next()	перейти к следующему элементу
v.prev()	перейти к предыдущему элементу
v.skip(n)	перейти на n элементов вперед
v.skip(-n)	перейти на n элементов назад
v.current()	получить текущий элемент

Итератор можно реализовать как класс, представляющий такой же набор операций. В C++ итератор реализуется как класс, который имеет такой же интерфейс, как и указатель для совместимости с массивами.

Если объект-итератор имеет имя iterv, то операции могут быть представлены следующим образом:

iterv=v.first()	перейти к первому элементу
iterv=v.last()	перейти к последнему элементу
iterv++	перейти к следующему элементу
iterv--	перейти к предыдущему элементу
iterv+=n	перейти на n элементов вперед
v.skip-=n	перейти на n элементов назад
*iterv	получить текущий элемент

При наличии последовательного доступа с помощью итератора обычными операциями являются операции:

- Вставки элемента перед текущим элементом (после текущего);
- Изменение значения текущего элемента;
- Удаление текущего элемента;
- Поиск элемента с заданным значением (возвращает итератор на текущий элемент).

2.2.2. Операции объединения контейнеров

Наиболее часто используется операция объединения двух контейнеров с получением нового контейнера. Она может быть реализована в разных вариантах:

- Простое сцепление двух контейнеров: в новый контейнер попадают сначала элементы первого контейнера, потом второго, операция не коммутативна.
- Объединение упорядоченных контейнеров, новый контейнер тоже будет упорядочен, операция коммутативна.
- Объединение контейнеров как объединение множеств, в новый контейнер попадают только те элементы, которые есть хотя бы в одном контейнере, операция коммутативна.
- Объединение контейнеров как пересечение множеств, в новый контейнер попадают только те элементы, которые есть в обоих контейнерах, операция коммутативна.
- Для контейнеров-множеств может быть еще реализована операция вычитания, в контейнер попадают только те элементы первого контейнера, которых нет во втором, операция не коммутативна.
- Извлечение части элементов из контейнера и создание нового контейнера. Эта операция может быть выполнена с помощью конструктора, а часть контейнера задается двумя итераторами.

3. 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.

4. Ход работы

Задача

Класс- контейнер ВЕКТОР с элементами типа int.
Реализовать операции:
[] – доступа по индексу;
() – определение размера вектора;
+ число – добавляет константу ко всем элементам вектора;
++ - переход к следующему элементу (с помощью класса-итератора).

1. Создать пустой проект.
2. Добавить в него класс Vector.
3. В файл Vector.h добавить описание класса Вектор:

```
class Vector
{
public:
//конструктор с параметрами: выделяет память под s элементов и заполняет их
//значением k
	Vector(int s,int k=0);
//конструктор с параметрами
	Vector(const Vector&a);
//деструктор
	~Vector();
//оператор присваивания
	Vector&operator=(const Vector&a);
//операция доступа по индексу
	int&operator[](int index);
//операция для добавление константы
	Vector operator+(const int k);
//операция, возвращающая длину вектора
	int operator()();
//перегруженные операции ввода-вывода
friend ostream& operator<<(ostream& out, const Vector&a);
friend istream& operator>>(istream& in, Vector&a);
private:
	int size;//размер вектора
	int*data;//укзатель на динамический массив значений вектора
};
```

4. В файл Vector.cpp добавить определение методов класса Вектор:

//конструктор с параметрами
```
Vector::Vector(int s,int k)
{
	size=s;
	data=new int[size];
	for(int i=0;i<size;i++)
		data[i]=k;
}
```
//конструктор копирования
```
Vector::Vector(const Vector&a)
{
	size=a.size;
	data=new int[size];
	for(int i=0;i<size;i++)
		data[i]=a.data[i];
}
```
//деструктор
```
Vector::~Vector()
{
	delete[]data;
	data=0;
```

```
}
//операция присваивания
Vector&Vector::operator=(const Vector&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];
	return *this;
}
//операция доступа по индексу
int&Vector::operator[](int index)
{
	if (index<size) return data[index];
	else  cout<<"\nError! Index>size";
}
//операция для добавления константы
Vector Vector::operator+(const int k)//+k
{
	Vector temp(size);
	for (int i=0;i<size;++i)
		temp.data[i]+=data[i]+k;
	return temp;
}
//операция для получения длины вектора
int Vector::operator ()()
{
	return len();
}
//операции для ввода-выода
ostream&operator<<(ostream&out,const Vector&a)
{
	for(int i=0;i<a.len();++i)
		out<<a.data[i]<<" ";
	return out;
}

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

5. Добавить в проект файл lab6_main().cpp с функцией main(), в которой выполнить тестирование класса Вектор.

```
void main()
{
	Vector a(5);//создали вектор из 5 элементов, заполненный нулями
	cout<<a<<"\n";//вывели значения элементов вектора
	cin>>a;//ввели с клавиатуры значения элементов вектора
	cout<<a<<"\n";//вывели значения элементов вектора
	a[2]=100;//используя операцию [] присвоили новое значение элементу
	cout<<a<<"\n";//вывели значения элементов вектора
	Vector b(10);//создали вектор b из 10 элементов, заполненный нулями
	cout<<b<<"\n";//вывели значения элементов вектора
	b=a;//присвоили вектору b значения вектора a
	cout<<b<<"\n";//вывели значения элементов вектора
	Vector c(10); //создали вектор c из 10 элементов, заполненный нулями
	c=b+100;//Увеличили значения вектора b на 100 и присвоили вектору c
	cout<<c<<"\n";//вывели значения элементов вектора c
	cout<<"\nthe length of a="<<a()<<endl;//вывели длину вектора a
```

```
}
```

6. В файл Vector.h добавить описание класса Итератор (перед классом Вектор):

```
class Iterator
{
	friend class Vector;//дружественный класс
public:
	Iterator(){elem=0;}//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;};
//перегруженная операция инкремент
	void operator++(){ ++elem;};
//перегруженная операция декремент
	void operator--(){--elem;}
//перегруженная операция разыменования
	int& operator *() const{ return*elem;}
private:
	int*elem;//указатель на элемент типа int
};
```

7. В класс Vector добавить атрибуты и методы для работы с итератором:

```
class Vector
{
public:

	. . . . . .
Iterator first(){return beg;}//возвращает указатель на первый элемент
Iterator last(){return end;}//возвращает указатель на элемент следующий за
//последним
private:
	int size;
	int*data;
	Iterator beg;//указатель на первый элемент вектора
	Iterator end;//указатель на элемент следующий за последним
};
```

8. В конструкторы добавить операторы, инициализирующие значения beg и end.

```
Vector::Vector(int s,int k)
{
	. . . . . .
	beg.elem=&data[0];
	end.elem=&data[size];
}


Vector::Vector(const Vector&a)
{
	. . . . . .
	beg=a.beg;
	end=a.end;
}
Vector&Vector::operator=(const Vector&a)
{
	. . . . . .
	beg=a.beg;
	end=a.end;
	return *this;
}
```

9. В функцию main() добавить операторы для тестирования операций итератора:

```
void main()
{
      . . . . . .
//разыменовываем значение, которое возвращает a.first() и выводим его
      cout<<*(a.first())<<endl;
//переменную типа Iterator устанавливаем на первый элемент вектора a с
//помощью метода first
      Iterator i=a.first();
//оперция инкремент
      i++;
//разыменовываеи итератор и выводи его значение
      cout<<*i<<endl;
//выводим значения элеменов вектора с помощью итератора
      for( i=a.first();i!=a.last();i++)cout<<*i<<endl;
}
```

5. Варианты

№	Задание
1	Класс- контейнер ВЕКТОР с элементами типа int. Реализовать операции: [] – доступа по индексу; () – определение размера вектора; + число – добавляет константу ко всем элементам вектора; ++ - переход к следующему элементу (с помощью класса-итератора).
2	Класс- контейнер ВЕКТОР с элементами типа int. Реализовать операции: []– доступа по индексу; int() – определение размера вектора; + вектор – сложение элементов векторов a[i]+b[i]; +n - переход вправо к элементу с номером n (с помощью класса-итератора).
3	Класс- контейнер ВЕКТОР с элементами типа int. Реализовать операции: [] – доступа по индексу; + вектор – сложение элементов векторов a[i]+b[i]; + число – добавляет константу ко всем элементам вектора; -- - переход к предыдущему элементу (с помощью класса-итератора).
4	Класс- контейнер ВЕКТОР с элементами типа int. Реализовать операции: [] – доступа по индексу; () – определение размера вектора; * число – умножает все элементы вектора на число; - n – переход влево к элементу с номером n (с помощью класса-итератора).
5	Класс- контейнер ВЕКТОР с элементами типа int. Реализовать операции: [] – доступа по индексу; int() – определение размера вектора;

	* вектор – умножение элементов векторов a[i]*b[i]; + n – переход вправо к элементу с номером n (с помощью класса-итератора).
6	Класс- контейнер МНОЖЕСТВО с элементами типа int. Реализовать операции: [] – доступа по индексу; () – определение размера множества; + – объединение множеств; ++ - переход к следующему элементу (с помощью класса-итератора).
7	Класс- контейнер МНОЖЕСТВО с элементами типа int. Реализовать операции: [] – доступа по индексу; int() – определение размера вектора; * – пересечение множеств; -- - переход к предыдущему элементу (с помощью класса-итератора).
8	Класс- контейнер МНОЖЕСТВО с элементами типа int. Реализовать операции: [] – доступа по индексу; == - проверка на равенство; > число – принадлежность числа множеству; - n - переход влево к элементу с номером n (с помощью класса-итератора).
9	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).
10	Класс- контейнер МНОЖЕСТВО с элементами типа int. Реализовать операции: [] – доступа по индексу; () – определение размера вектора; - – разность множеств; -- – переход к предыдущему элементу (с помощью класса-итератора).
11	Класс- контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции: []– доступа по индексу; int() – определение размера списка; + вектор – сложение элементов списков a[i]+b[i]; - n - переход влево к элементу с номером n (с помощью класса-итератора).

12	Класс- контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции: [] – доступа по индексу; () – определение размера вектора; + число – добавляет константу ко всем элементам вектора; ++ - переход к следующему элементу (с помощью класса-итератора).
13	Класс- контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции: [] – доступа по индексу; + вектор – сложение элементов списков a[i]+b[i]; + число – добавляет константу ко всем элементам списка; -- - переход к предыдущему элементу (с помощью класса-итератора).
14	Класс- контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции: [] – доступа по индексу; () – определение размера списка; * число – умножает все элементы списка на число; - n – переход влево к элементу с номером n (с помощью класса-итератора).
15	Класс- контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции: [] – доступа по индексу; int() – определение размера списка; * вектор – умножение элементов списков a[i]*b[i]; +n - переход вправо к элементу с номером n (с помощью класса-итератора).

6. Контрольные вопросы

1. Что такое абстрактный тип данных? Привести примеры АТД.
2. Привести примеры абстракции через параметризацию.
3. Привести примеры абстракции через спецификацию.
4. Что такое контейнер? Привести примеры.
5. Какие группы операций выделяют в контейнерах?
6. Какие виды доступа к элементам контейнера существуют? Привести примеры.
7. Что такое итератор?
8. Каким образом может быть реализован итератор?
9. Каким образом можно организовать объединение контейнеров?
10. Какой доступ к элементам предоставляет контейнер, состоящий из элементов «ключ-значение»?
11. Как называется контейнер, в котором вставка и удаление элементов выполняется на одном конце контейнера?
12. Какой из объектов (a,b,c,d) является контейнером?
    a. int mas=10;
    b. 2. int mas;
    c. 3. struct {char name[30]; int age;} mas;
    d. 4. int mas[100];
13. Какой из объектов (a,b,c,d) не является контейнером?
    a. int a[]={1,2,3,4,5};

b. 2. int mas[30];
c. 3. struct {char name[30]; int age;} mas[30];
d. 4. int mas;

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

7. Содержание отчета

1) Постановка задачи (общая и конкретного варианта).
2) Описание класса-контейнера.
3) Определение компонентных функций.
4) Описание класса-итератора и его компонентных функций
5) Функция main().
6) Объяснение результатов работы программы.
7) Ответы на контрольные вопросы.