**Лабораторная работа № 3. “ПЕРЕГРУЗКА ОПЕРАЦИЙ”**  
**Цель.** Получить практические навыки создания абстрактных типов данных и перегрузки операций.  
**Основное содержание работы.**  
Определить и реализовать класс − абстрактный тип данных. Определить и реализовать операции над данными этого класса.  
**Краткие теоретические сведения.**  
**Абстрактный тип данных (АТД).**  
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 С++.  
**Перегрузка операций.**  
Возможность использовать знаки стандартных операций для записи выражений как для встроенных, так и для АТД.  
В языке С++ для перегрузки операций используется ключевое слово operator, с помощью которого определяется специальная операция-функция (operator function).

Перегрузка операций.  
Возможность использовать знаки стандартных операций для записи выражений как для встроенных, так и для АТД. В языке С++ для перегрузки операций используется ключевое слово  
operator, с помощью которого определяется специальная операция-функция (operator function).  
Формат операции-функции:  
тип\_возвр\_значения operator знак\_операции (специф\_параметров)  
{операторы\_тела\_функции}  
Перегрузка унарных операций

• Любая унарная операция ⊕ может быть определена двумя способами: либо как компонентная функция без параметров, либо как глобальная (возможно дружественная) функция с одним параметром. В первом случае выражение ⊕ Z означает вызов Z.operator ⊕ (), во втором − вызов  
operator ⊕(Z).

• Унарные операции, перегружаемые в рамках определенного класса, могут перегружаться только через нестатическую компонентную функцию без параметров. Вызываемый объект класса автоматически воспринимается как операнд.  
• Унарные операции, перегружаемые вне области класса (как глобальные функции), должны иметь один параметр типа класса. Передаваемый через этот параметр объект воспринимается как операнд.

Синтаксис:  
а) в первом случае (описание в области класса):  
тип\_возвр\_значения operator знак\_операции  
б) во втором случае (описание вне области класса):  
тип\_возвр\_значения operator знак\_операции(идентификатор\_типа)

Пример 1.

C++:

1)

class Person

{

int age;

...

public:

…

void operator++(){ ++age;}

};

void main()

{

Person jon;

++jon;

}

2)

class Person

{

int age;

...

public:

...

friend void operator++(Person&);

};

void person::operator++(Person& ob)

{

++ob.age;

}

void main()

{

Person jon;  
++jon;

}

Примеры на других языках находятся в приложении А.

Пояснение: в данном примере продемонстрирована перегрузка унарной операции ++. После данной перегрузки, после применения к объекту класса Person ++, атрибуту age будет инкрементирована единица.

**Перегрузка бинарных операций**  
• Любая бинарная операция ⊕ может быть определена двумя способами: либо как компонентная функция с одним параметром, либо как глобальная (возможно дружественная) функция с двумя параметрами. В первом случае x⊕y означает вызов x.operator⊕(y), во втором – вызов operator⊕(x,y).  
• Операции, перегружаемые внутри класса, могут перегружаться только нестатическими компонентными функциями с параметрами. Вызываемый объект класса автоматически воспринимается в качестве первого операнда.  
• Операции, перегружаемые вне области класса, должны иметь два операнда, один из которых должен иметь тип класса.  
Пример 2 C++.  
1)

class Person{…};  
class Adresbook  
{

// содержит в качестве компонентных данных множество объектов типа

//person, представляемых как динамический массив, список или дерево  
…  
public:  
Person& operator[](int); //доступ к i-му объекту  
};  
Person& Adresbook : : operator[](int i){. . .}  
void main()  
{

Adresbook persons;  
Person record;  
…  
record = persons [3];

}  
2)

class Person{…};  
class Adresbook  
{ // содержит в качестве компонентных данных множество объектов типа

//person, представляемых как динамический массив, список или дерево  
…  
public:  
friend Person& operator[](const Adresbook&,int); //доступ к i-му объекту  
};  
Person& operator[](const Adresbook& ob ,int i){. . .}  
void main()  
{

Adresbook persons;  
Person record;  
…  
record = persons [3];

}

Перегрузка операции присваивания  
Операция отличается тремя особенностями:  
• операция не наследуется;

• операция определена по умолчанию для каждого класса в качестве операции поразрядного копирования объекта, стоящего справа от знака операции, в объект, стоящий слева.  
• операция может перегружаться только в области определения  
класса. Это гарантирует, что первым операндом всегда будет лево допустимое выражение.  
Формат перегруженной операции присваивания:  
имя\_класса& operator=( имя\_класса&);  
Отметим две важные особенности функции operanor=. Во-первых, в ней используется параметр-ссылка. Это необходимо для предотвращения создания копии объекта, передаваемого через параметр по значению. В случаи создания копии, она удаляется вызовом деструктора при завершении работы функции. Но деструктор освобождает распределенную память,  
еще необходимую объекту, который является аргументом. Параметр-ссылка помогает решить эту проблему.  
Во-вторых, функция operator=() возвращает не объект, а ссылку на  
него. Смысл этого тот же, что и при использовании параметра-ссылки.  
Функция возвращает временный объект, который удаляется после завершения ее работы. Это означает, что для временной переменной будет вызван деструктор, который освобождает распределенную память. Но она необходима для присваивания значения объекту. Поэтому, чтобы избежать создания временного объекта, в качестве возвращаемого значения используется ссылка.

**Порядок выполнения работы.**  
1. Выбрать класс АТД в соответствии с вариантом.  
2. Определить и реализовать в классе конструкторы, деструктор,  
функции Input (ввод с клавиатуры) и Print (вывод на экран), перегрузить  
операцию присваивания.  
3. Написать программу тестирования класса и выполнить тестирование.  
4. Дополнить определение класса заданными перегруженными операциями ( в соответствии с вариантом).  
5. Реализовать эти операции. Выполнить тестирование.

6. Реализовать вариант на языке С++ и (С#/Java на выбор).

**Методические указания.**  
1.Класс АТД реализовать как динамический массив. Для этого определение класса должно иметь следующие поля:  
− указатель на начало массива;  
− максимальный размер массива;  
− текущий размер массива.  
2. Конструкторы класса размещают массив в памяти и устанавливают его максимальный и текущий размер. Для задания максимального массива использовать константу, определяемую вне класса.  
3. Чтобы у вас не возникало проблем, аккуратно работайте с константными объектами. Например:  
•конструктор копирования следует определить так:  
MyClass (**const** MyClass& ob);  
•операцию присваивания перегрузить так:  
MyClass& operator = (**const** MyClass& ob);

4. Для удобства реализации операций-функций реализовать в классе  
**private(protected**)-функции, работающие непосредственно с реализацией  
класса. Например, для класса **множество** это могут быть следующие  
функции:

− включить элемент в множество;  
− найти элемент и возвратить его индекс;  
− удалить элемент;  
− определить, принадлежит ли элемент множеству.  
Указанные функции используются в реализации общедоступных  
функций-операций (operator).

**Содержание отчета.**  
1. Титульный лист.  
2. Конкретное задание с указанием номера варианта, реализуемого  
класса и операций.  
3. Определение класса.  
4. Обоснование включения в класс нескольких конструкторов, дест-  
руктора и операции присваивания.  
5. Объяснить выбранное представление памяти для объектов реали-  
зуемого класса.  
7. Тестовые данные и результаты тестирования.  
**Вопросы для самоконтроля.**  
1. Что такое абстрактный тип данных?  
2. Приведите примеры абстрактных типов данных.  
3. Каковы синтаксис/семантика “операции-функции”?  
4. Как можно вызвать операцию-функцию?  
5. Нужно ли перегружать операцию присваивания относительно оп-  
ределенного пользователем типа данных, например класса? Почему?  
6. Можно ли изменить приоритет перегруженной операции?  
7. Можно ли изменить количество операндов перегруженной опера-  
ции?  
8. Можно ли изменить ассоциативность перегруженной операции?  
9. Можно ли, используя дружественную функцию, перегрузить опе-  
ратор присваивания?  
10. Все ли операторы языка С++ могут быть перегружены?  
11. Какими двумя разными способами определяются перегруженные  
операции?  
12. Все ли операции можно перегрузить с помощью глобальной дру-  
жественной функции?

13. В каких случаях операцию можно перегрузить только глобальной  
функцией?  
14. В каких случаях глобальная операция-функция должна быть  
дружественной?  
15. Обязателен ли в функции operator параметр типа “класс” или  
“ссылка на класс”?  
16. Наследуются ли перегруженные операции?  
17. Можно ли повторно перегрузить в производном классе опера-  
цию, перегруженную в базовом классе?  
18. В чем отличие синтаксиса операции-функции унарной и бинар-  
ной операции?  
19. Приведите примеры перегрузки операций для стандартных ти-  
пов.  
20. Перегрузите операцию “+” для класса “комплексное число”.  
21. Перегрузите операции “<”,”>”,”==” для класса “строка символов.

**Приложение. Варианты заданий.**  
1. АТД − множество с элементами типа **char.** Дополнительно перегрузить следующие операции:  
+ − добавить элемент в множество(типа char + set);  
+ − объединение множеств;  
== − проверка множеств на равенство.  
2. АТД − множество с элементами типа **char.** Дополнительно перегрузить следующие операции:  
- − удалить элемент из множества (типа set-char);  
\* − пересечение множеств;  
< − сравнение множеств.  
3. АТД − множество с элементами типа **char.** Дополнительно перегрузить следующие операции:  
- − удалить элемент из множества (типа set-char);  
> − проверка на подмножество;  
!= − проверка множеств на неравенство.  
4. АТД − множество с элементами типа **char.** Дополнительно перегрузить следующие операции:  
+ − добавить элемент в множество (типа set+char);  
\* −пересечение множеств;

int()− мощность множества.  
5. АТД − множество с элементами типа **char.** Дополнительно перегрузить следующие операции:  
() − конструктор множества (в стиле конструктора Паскаля);  
+ − объединение множеств;  
<= − сравнение множеств .  
6. АТД − множество с элементами типа **char.** Дополнительно перегрузить следующие операции:  
> − проверка на принадлежность(char in set Паскаля);  
\* − пересечение множеств;  
< − проверка на подмножество.  
7. АТД − однонаправленный список с элементами типа **char.** Дополнительно перегрузить следующие операции:  
+ – объединить списки (list+list);  
-- – удалить элемент из начала (типа --list);  
= = – проверка на равенство.  
**8. АТД − однонаправленный список с элементами типа char. Дополнительно перегрузить следующие операции:  
+ – добавить элемент в начало(char+list);  
-- – удалить элемент из начала(типа –list);  
= = – проверка на равенство.**9. АТД − однонаправленный список с элементами типа **char.** Дополнительно перегрузить следующие операции:  
+ − добавить элемент в конец (list+char);  
-- − удалить элемент из конца (типа list--);  
!= − проверка на неравенство.  
10. АТД − однонаправленный список с элементами типа **char.** Дополнительно перегрузить следующие операции:  
[] − доступ к элементу в заданной позиции, например:  
int i; char c;  
list L;  
c=L[i];  
+ − объединить два списка;

== − проверка на равенство.

11. АДТ − однонаправленный список с элементами типа **char.** Дополнительно перегрузить следующие операции:  
[] − доступ к элементу в заданной позиции, например:  
int i; char c;  
list L;  
c=L[i];  
+ − объединить два списка;  
!= − проверка на неравенство.

12. АДТ − однонаправленный список с элементами типа **char.** Дополнительно перегрузить следующие операции:  
() − удалить элемент в заданной позиции, например :  
int i;  
list L;  
L[i];  
() − добавить элемент в заданную позицию, например :  
int i; char c;  
list L;  
L[с,i];  
!= − проверка на неравенство.

13. АДТ − стек. Дополнительно перегрузить следующие операции:  
+ − добавить элемент в стек;  
− − извлечь элемент из стека;  
bool()− проверка, пустой ли стек.  
14. АДТ − очередь. Дополнительно перегрузить следующие операции:  
+ − добавить элемент;  
− − извлечь элемент;  
bool() – проверка, пустая ли очередь.

15. АДТ − одномерный массив (вектор) вещественных чисел. Дополнительно перегрузить следующие операции:  
+ − сложение векторов (a[i]+b[i] для всех i);  
[] − доступ по индексу;  
+ − добавить число к вектору (double+vector).  
16. АТД − одномерный массив (вектор) вещественных чисел. Дополнительно перегрузить следующие операции:

- − вычитание векторов (a[i]-b[i] для всех i);  
[] − доступ по индексу;  
- − вычесть из вектора число (vector-double).  
17. АТД − одномерный массив (вектор) вещественных чисел. Дополнительно перегрузить следующие операции:  
\* − умножение векторов (a[i]\*b[i] для всех i);  
[] − доступ по индексу;  
\* − умножить вектор на число (vector\*double).  
18. АТД − одномерный массив (вектор) вещественных чисел. Дополнительно перегрузить следующие операции:  
int() − размер вектора;  
() − установить новый размер;  
- − вычесть из вектора число (vector-double);  
[] − доступ по индексу;  
19. АТД − одномерный массив (вектор) вещественных чисел. Дополнительно перегрузить следующие операции:  
= − присвоить всем элементам вектора значение (vector=double);  
[] − доступ по индексу;  
= = − проверка на равенство;  
!= − проверка на неравенство;

20. АТД − двухмерный массив (матрица) вещественных чисел. Дополнительно перегрузить следующие операции:  
() − доступ по индексу;  
\* − умножение матриц;  
\* − умножение матрицы на число;  
\* − умножение числа на матрицу.  
21. АТД − двухмерный массив (матрица) вещественных чисел. Дополнительно перегрузить следующие операции:  
() − доступ по индексу;  
- − разность матриц;  
- − вычесть из матрицы число;  
== − проверка матриц на равенство.  
22. АТД − двухмерный массив (матрица) вещественных чисел. Дополнительно перегрузить следующие операции:  
() − доступ по индексу;

= − присвоить всем элементам матрицы значение (matr=double);

+ − сложение матриц;

+ − сложить матрицу с числом (matr+double).  
23. АТД − двухмерный массив (матрица) вещественных чисел. Дополнительно перегрузить следующие операции:  
() − доступ по индексу;  
== − проверка матриц на равенство;  
++ − транспонировать матрицу.

**Приложение А.**

**Пример 1 (С#):**

class Person

{

private int age;

public int getAge() { return age; }

public void setAge(int age) {this.age = age;}

public static Person operator ++ (Person obj)

{

obj.age++;

return obj;

}

}

class Program

{

static void Main(string[] args)

{

Person ss = new Person();

ss.setAge(4);

ss++;

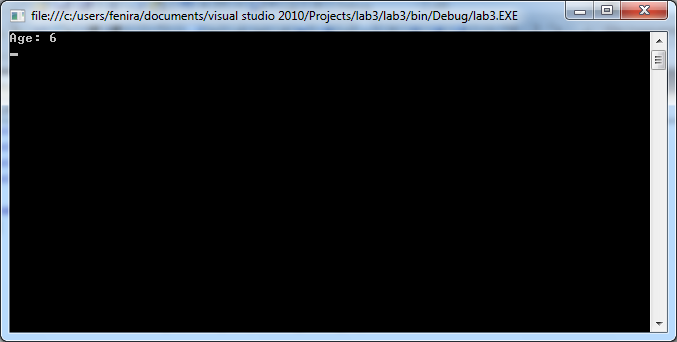
++ss;

Console.WriteLine("Age: {0}", ss.getAge());

Console.ReadKey();

}

}



Пример перегрузки бинарных операций С#:

class Complex

{

private int real;

private int imaginary;

public Complex()

{

this.real = 0;

this.imaginary = 0;

}

public Complex(int real, int imaginary)

{

this.real = real;

this.imaginary = imaginary;

}

public void setComplex(int real, int imaginary)

{

this.real = real;

this.imaginary = imaginary;

}

public void printComplex()

{

if (imaginary >= 0)

{

Console.WriteLine("{0}+{1}i", real, imaginary);

}

else

{

Console.WriteLine("{0}{1}i", real, imaginary);

}

}

public static Complex operator +(Complex obj1, Complex obj2)

{

Complex obj3 = new Complex();

obj3.real = obj1.real + obj2.real;

obj3.imaginary = obj1.imaginary + obj2.imaginary;

return obj3;

}

}

class Program

{

static void Main(string[] args)

{

Complex com1 = new Complex(1, -4);

com1.printComplex();

Complex com2 = new Complex(3, 6);

com2.printComplex();

Complex com3 = com1 + com2;

com3.printComplex();

Console.ReadKey();

}

}

