# Лабораторная работа №2

**Тема работы:** дружественные функции и классы, перегрузка операторов.

**Цель работы:** Понять назначение дружественных функций и классов, изучить принципы перегрузки бинарных и унарных операций.

**Теоретические сведения:**

**Дружественные функции.** Иногда возникает необходимость организации доступа к локальным данным нескольких классов из одной функции. Для реализации этого в С++ введен спецификатор friend. Если некоторая функция определена как friend-функция для некоторого класса, то эта функция называется дружественной и она:

* не является компонентом-функцией этого класса;
* имеет доступ ко всем компонентам этого класса (private, public и protected).

Пример использования дружественной функции.

#include <iostream>

using namespace std;

class kls

{

int i,j;

public:

kls(int I,int J) : i(I),j(J) {} // конструктор

int max() {return i>j? i : j;} // функция-компонент класса kls

friend double fun(int, kls&); // friend-объявление внешней функции fun

};

double fun(int i, kls &x) // описание дружественной функции

{

return (double)i/x.i;

}

int main()

{

kls obj(2,3); // объявление объекта

cout << obj.max() << endl;

cout << fun(3,obj) << endl; // вызов дружественной функции

return 0;

}

Функции со спецификатором friend, не являясь компонентами класса, не имеют и, следовательно, не могут использовать this указатель. Следует также отметить ошибочность следующей заголовочной записи функции

double kls :: fun(int i,int j),

т. к. fun не является компонентом-функцией класса kls.

В общем случае friend-функция является глобальной независимо от секции, в которой она объявлена (public, protected, private), при условии, что она не объявлена ни в одном другом классе без спецификатора friend. Функция friend, объявленная в классе, может рассматриваться как часть интерфейса класса с внешней средой.

Вызов компоненты-функции класса осуществляется с использованием операции доступа к компоненте «.» или «->». Вызов же friend-функции производится по ее имени. В friend-функцию не передается this-указатель и доступ к компонентам класса выполняется либо явно «.», либо косвенно «->».

Компонент-функция одного класса может быть объявлена со спецификатором friend для другого класса:

class X{ ……….

void fun (….);

};

class Y{ ……….

friend void X:: fun (….);

};

В приведенном фрагменте функция fun() имеет доступ к локальным компонентам класса Y. Запись вида friend void X:: fun (….) говорит о том, что функция fun принадлежит классу Х, а спецификатор friend разрешает доступ к локальным компонентам класса Y (т. к. она объявлена со спецификатором в классе Y).

Ниже приведен пример программы расчета суммы двух налогов на зарплату. Используется дружественная функция.

#include <iostream>

#include "string.h"

#include <iomanip>

using namespace std;

class nalogi; // неполное объявление класса nalogi

class work

{

char s[20]; // фамилия работника

int zp; // зарплата

public:

float raschet(nalogi); // компонент-функция класса work

void inpt()

{

cout << "вводите фамилию и зарплату" << endl;

cin >> s >> zp;

}

work(){}

~work(){};

};

class nalogi

{

float pd, // подоходный налог

st; // налог на социальное страхование

friend float work::raschet(nalogi); // friend-функция класса nalogi

public:

nalogi(float f1,float f2) : pd(f1),st(f2){};

~nalogi(void){};

};

float work::raschet(nalogi nl)

{

cout << s << setw(6) << zp <<endl; // доступ к данным класса work

cout << nl.pd << setw(8) << nl.st <<endl; // доступ к данным класса nalogi

return zp\*nl.pd/100+zp\*nl.st/100;

}

int main()

{

setlocale(LC\_ALL,"Russian");

nalogi nlg((float)12,(float)2.3); // описание и инициализация объекта

work wr[2]; // описание массива объектов

for(int i=0;i<2;i++)

wr[i].inpt(); // инициализация массива объктов

cout << setiosflags(ios::fixed) << setprecision(3) << endl;

cout << wr[0].raschet(nlg) << endl; // расчет налога для объекта wr[0]

cout << wr[1].raschet(nlg) << endl; // расчет налога для объекта wr[1]

return 0;

}

Следует отметить необходимость выполнения неполного (предварительного) объявления класса nalogi, т. к. в прототипе функции raschet класса work используется объект класса nalogi, объявляемого далее. Все функции одного класса можно объявить со спецификатором friend по отношению к другому классу следующим образом.

class X{ ………..

friend class Y;

………..

};

class Y{ ……….

};

В этом случае все компоненты-функции класса Y имеют спецификатор friend для класса Х (имеют доступ к компонентам класса Х). Класс Y является дружественным классу X.

В приведенном ниже примере оба класса являются дружественными.

#include <iostream>

using namespace std;

class A

{

int i; // компонент-данное класса А

public:

friend class B; // объявление класса В другом класса А

A():i(1){} // конструктор

~A(){} // деструктор

void f1\_A(B &); // метод, оперирующий данными обоих классов

};

class B

{

int j; // компонент-данное класса В

public:

friend A; // объявление класса А другом класса В

B():j(2){} // конструктор

~B(){} // деструктор

void f1\_B(A &a) // метод, оперирующий данными обоих классов

{

cout << a.i + j << endl;

}

};

void A :: f1\_A(B &b)

{

cout << i << ' ' << b.j<< endl;

}

int main()

{

A aa;

B bb;

aa.f1\_A(bb);

bb.f1\_B(aa);

}

Результат выполнения программы:

1 2

3

В объявлении класса А содержится инструкция friend class B, являющаяся и предварительным объявлением класса В и объявлением класса В дружественным классу А. Отметим также необходимость описания функции f1\_A после явного объявления класса В (в противном случае не может быть создан объект b еще не объявленного типа).

Отметим основные свойства и правила использования спецификатора friend:

* friend-функции не являются компонентами класса, но имеют доступ ко всем его компонентам независимо от их атрибута доступа;
* friend-функции не имеют указателя this;
* friend-функции не наследуются в производных классах;
* отношение friend не является *ни симметричным* (т. е. если класс А friend классу В, то это не означает, что В также является friend классу А), *ни транзитивным* (т. е. если A friend B и B friend C, то не следует, что A friend C);
* друзьями класса можно определить перегруженные функции. Каждая перегруженная функция, используемая как friend для некоторого класса, должна быть явно объявлена в классе со спецификатором friend.

**Перегрузка операторов.** Программы на языке С++ используют некоторые ранее определенные простейшие классы (типы), такие, как int, char, float и т. д. Мы можем описать объекты указанных классов, например:

int a,b;

char c,d,e;

float f;

Здесь переменные a, b, c, d, e, f можно рассматривать как простейшие объекты. В языке определены множество операций над простейшими объектами, выражаемых через операторы, такие, как «+», «-», «\*», «/», «%» и т. д. Каждый оператор можно применить к операндам определенного типа.

float a, b=3.123, c=6.871;

a=c+b; // нет ошибки

a=c%b; // ошибка

Второе является ошибочным, поскольку оператор «%» должен быть приложен лишь к объектам целого типа. Из этого следует, что операторы языка можно применить к тем объектам, для которых они были определены.

К сожалению, лишь определенное число типов непосредственно поддерживается любым языком программирования. Однако многие операторы можно определить через классы в С++. Рассмотрим пример.

Пусть заданы множества А и В:

А={а1,а2,а3}; B={a3,a4,a5};

Необходимо выполнить операции пересечение множеств «&» и объединение множеств «|»:

A&B={a1,a2,a3} & {a3,a4,a5}={a3};

A|B={a1,a2,a3} | {a3,a4,a5}={a1,a2,a3a4,a5};

Языки С/С++ не поддерживают непосредственно эти операции, однако в языке С++ можно объявить класс, назвав его set (множество). Далее можно определить операции над этим классом, выразив их с помощью знаков операторов, которые уже есть в языке С++, например «&» и «|». В результате операции «&» и «|» можно будет использовать как и раньше, а также снабдить их дополнительными функциями (объединения и пересечения множеств). Как определить, какую функцию должен выполнять оператор: старую или новую? Надо посмотреть на тип операндов в соответствующем выражении. Если операнды – это объекты целого типа, то нужно выполнить операцию «побитового И» или «побитового ИЛИ». Если же операнды – это объекты типа set, то надо выполнить объединение или пересечение соответствующих множеств.

**Функция operator.** Функция operator может быть использована для расширения области приложения следующих операторов: «+», «-», «\*», «%», «&», «|» и т.д. Операторы, которые не могут быть перегружены: «.», «.\*», «::», «?:», sizeof и typeid.

Для перегрузки (доопределения) оператора разрабатываются функции, являющиеся либо компонентами, либо friend-функциями того класса, для которого они используются. Для того, чтобы перегрузить оператор, требуется определить действие этой оператора внутри класса. Общая форма записи функции-оператора являющейся компонентой класса, имеет вид

тип\_возв\_значения имя\_класса::operator#(список аргументов)

{

действия, выполняемые применительно к классу

}

Вместо символа «#» ставится значок перегружаемого оператора. Оператор всегда определяется по отношению к компонентам некоторого класса. В результате ее старое предназначение сохраняет силу. Функция operator является компонентом класса. При этом в случае унарного оператора функция operator не будет иметь аргументов, а в случае бинарного оператора будет иметь один аргумент. В качестве отсутствующего аргумента используется указатель this на тот объект, в котором определен оператор. Объявление и вызов функции operator осуществляется так же, как и любой другой функции. Единственное ее отличие заключается в том, что разрешается использовать сокращенную форму ее вызова. Так, выражение operator#(a,b), можно записать в сокращенной форме a#b.

Рассмотрим пример. Программа доопределяет значение оператора «&». В результате ее можно будет использовать для выполнения операции пересечения множеств.

#include<iostream>

using namespace std;

class set // класс «множество»

{

char str[80]; // строка

public:

set(char \*ss) // это конструктор

{

strcpy(str,ss);

}

char \* operator&(set); // объявление функции operator

};

char \* set::operator&(set S) // описание функции operator

{

int t=0, l=0;

while(str[t++]!='\0'); // вычисление длины строки

char \*s1=new char[t]; // выделение памяти под строку

for(int j=0; str[j]!='\0'; j++)

for(int k=0; S.str[k]!='\0'; k++)

if(str[j]==S.str[k])

{

s1[l]=str[j];

l++;

break;

}

s1[l]='\0';

return s1;

}

int main()

{

set S1="1f2bg5e6", S2="abcdef"; // задаются два множества

cout << (S1 & S2) << endl; // результат fbe

cout << (set("123") & set("426")) << endl; // результат 2

}

Ниже приведен пример программы перегрузки унарного оператора.

#include <iostream>

using namespace std;

class dek\_koord

{

int x,y; // декартовы координаты точки

public:

dek\_koord(){}; // конструктор без параметров

dek\_koord(int X,int Y) // конструктор с параметрами

{

x=X;

y=Y;

}

void operator++(); // перегрузка префиксного оператора ++

void operator++(int); // перегрузка постфиксного оператора ++

dek\_koord operator=(dek\_koord); // перегрузка оператора =

void see();

};

void dek\_koord::operator++() // определение перегрузки оператора ++A

{

x++;

}

void dek\_koord::operator++(int) // определение перегрузки оператора A++

{

y++;

}

dek\_koord dek\_koord::operator =(dek\_koord a)

{

x=a.x; // определение перегрузки оператора =

y=a.y;

return \*this;

}

void dek\_koord::see()

{

cout << "координата х = " << x << endl;

cout << "координата y = " << y << endl;

}

int main()

{

setlocale(LC\_ALL,"Russian");

dek\_koord A(1,2), B, C; // объявление объектов

A.see();

A++; // увеличение значения компонента х объекта А

A.see(); // просмотр содержимого объекта А

++A; // увеличение значения компонента у объекта А

A.see(); // просмотр содержимого объекта А

C=B=A; // множественное присваивание

B.see();

C.see();

return 0;

}

**Контрольные вопросы**

1. Почему может потребоваться перегрузка оператора присваивания?
2. Можно ли изменить приоритет перегруженного оператора?
3. Когда следует переопределять операторы с помощью дружественных функций, а когда с помощью функций элементов класса?
4. Назовите особенности дружественных функций.
5. Опишите особенности перегрузки постфиксных и префиксных операторов «++» и «--».

**Порядок выполнения работы**

1. Изучить краткие теоретические сведения.
2. Ознакомиться с материалами литературных источников.
3. Ответить на контрольные вопросы.
4. Разработать алгоритм программы.
5. Написать, отладить и выполнить программу.

**Варианты заданий**

1. Реализовать класс String для работы со строками символов. Перегрузить операторов «=», «+=», «==», «<», «>», «!=», «<=», «>=». Предоставить конструктор копирования. Определить friend-функции для операторов ввода/вывода в поток.
2. Реализовать класс String для работы со строками символов. Перегрузить для объектов класса String операторы «!» (пуст ли объект класса), «()(int,int)» (возвращение подстроки объекта), «[]» (возвращение некоторого символа строки объекта). Предоставить конструктор копирования. Определить friend-функции для операторов ввода/вывода в поток.
3. Создать класс, в котором перегрузить операторы:

а) «&» для перемножения двух матриц;

б) «+» для сложения двух матриц.

Память под матрицы отводить динамически. Предоставить конструктор копирования. Определить friend-функции для операторов ввода/вывода в поток.

1. Реализовать класс String для работы со строками символов. Перегрузить операторы «=», «+=» так, чтобы производилось сложение строки и объекта. Предоставить конструктор копирования. Определить friend-функции для операторов ввода/вывода в поток.
2. Создать класс bMoney, в котором необходимо перегрузить арифметические операторы для работы с денежным форматом. Перегрузить два оператора следующим образом:

long double \* bMoney // умножить число на деньги

long double / bMoney // делить число на деньги

Эти операторы требуют наличия дружественных функций, т. к. справа от оператора находится объект, а слева – обычное число. Необходимо убедиться, что main() позволяет пользователю ввести две денежные строки и число с плавающей запятой, а затем корректно выполнить все семь арифметических действий с соответствующими парами значений.

1. Написать функцию инкремента единственного параметра. Написать функцию, возвращающую ссылку на передаваемый параметр. Изменить его при вызове функции. Что произойдет, если все ссылки сделать const?
2. Написать функцию декремента единственного параметра. Написать функцию, возвращающую ссылку на элемент глобального массива. Изменить его при вызове функции. Что произойдет, если все ссылки сделать const?
3. Реализовать класс String для работы со строками символов. Перегрузить операторы «=», «+=» так, чтобы производилось сложение строки и объекта. Предоставить конструктор копирования. Определить friend-функции для операторов ввода/вывода в поток.
4. Реализовать класс String для работы со строками символов. Перегрузить операторы «+» (сложение строк), «>» ( сравнение строк). Предоставить конструктор копирования.
5. Реализовать класс String для работы со строками символов. Перегрузить оператор «-» (минус) так, чтобы определить, насколько одна строка длиннее другой. Предоставить конструктор копирования.
6. Реализовать класс String для работы со строками символов. Перегрузить оператор «>», так, чтобы возвратить разность кодов первой пары несовпадающих символов в строках. Предоставить конструктор копирования.
7. Создать класс, в котором перегрузить оператор «&» для пересечения двух множеств. Память под матрицы отводить динамически. Предоставить конструктор копирования.
8. Создать класс, в котором перегрузить оператор «+» для объединения двух множеств. Память под матрицы отводить динамически. Предоставить конструктор копирования.
9. Реализовать класс String для работы со строками символов. Перегрузить унарные операторы «++» (префиксную и постфиксную). Предоставить конструктор копирования.
10. Создать два класса: вектор и матрица. Определить конструкторы (по умолчанию, с параметрами, копирования), деструкторы. Определить функцию умножения матрицы на вектор как дружественную.