**Работа № 7.  
Множественное наследование с использованием абстрактных  базовых классов, файлового ввода-вывода с применением потоков С++,  функций обработки исключительных ситуаций**

**Теоретические сведения**

**Задание к работе**

**Варианты заданий**

**Цель работы –** изучить методику создания множественного наследования, использование абстрактного базового класса, файловый ввод – вывод и использование функций обработки исключительных ситуаций.

## Теоретические сведения

Абстрактные классы

Если базовый класс используется только для порождения производных классов, то виртуальные функции в базовом классе могут быть "пустыми", поскольку никогда не будут вызваны для объекта базового класса. Базовый класс в котором есть хотя бы одна такая функция, называется абстрактным. Виртуальные функции в определении класса обозначаются следующим образом:

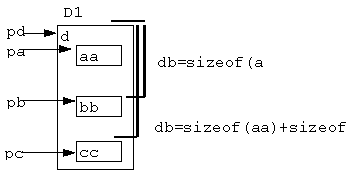
class base  
{  
public:  
virtual print()=0;  
virtual get() =0;  
};

Определять тела этих функций не требуется.

Множественное наследование

Множественным наследованием называется процесс создания производного класса из двух и более базовых. В этом случае производный класс наследует данные и функции всех своих базовых предшественников. Существенным для реализации множественного наследования является то, что адреса объектов второго и последующих базовых классов не совпадают с адресом объекта производного класса. Этот факт должен учитываться транслятором при преобразовании указателя на производный класс в указатель на базовый и наоборот:

class d : public a,public b, public c { };  
d       D1;  
pd =    &D1;        // #define db sizeof(a)  
pa =    pd;         // #define dc sizeof(a)+sizeof(b)  
pb =    pd;         // pb = (char\*)pd + db  
pc =    pd;         // pc = (char\*)pd + dc



Такое действие выполняется компилятором как явно при преобразовании в программе типов указателей, так и неявно, когда в объекте производного класса наследуется функция из второго и последующих базовых классов. Для вышеуказанного примера при определении в классе bb функции f() и ее наследовании в классе "d" вызов D1.f() будет реализован следующим образом:

this = &D1;                 // Указатель на объект производного класса  
this = (char\*)this + db     // Смещение к объекту базового класса  
b::f(this);                 // Вызов функции в базовом классе

Механизм виртуальных функций при множественном наследовании имеет свои особенности. Во-первых, на каждый базовый класс в производном классе создается свой массив виртуальных функций (в нашем случае -для aa в d, для bb в d и для cc в d). Во-вторых, если функция базового класса переопределена в производном, то при ее вызове требуется преобразовать указатель на объект базового класса в указатель на объект производного. Для этого транслятор включает соответствующий код, корректирующий значение this в виде "заплаты", передающей управление командой перехода к переопределяемой функции, либо создает отдельные таблицы смещений.

Файловые потоки. Классы файловых потоков:

ifstream   - файл ввода, производный от istream

ofstream   - файл вывода, производный от ostream

fstream     - файл ввода-вывода, производный от iostream

Флаги режимов работы с файлом:

enum ios::open\_mode  
{  
in = 0x01,         // Открыть файл только для чтения  
out =    0x02,         // Открыть файл только для записи  
ate =    0x04,         // При открытии позиционироваться в конец файла   
app =    0x08,         // Открыть существующий для дополнения  
trunc =  0x10,    // Создание нового файла взамен существующего   
nocreate=0x20,     // Не создавать новый файл при его отсутствии  
noreplace=0x40,    // Не создавать новый файл, если он существует  
binary=  0x80 // Двоичный файл ("прозрачный" ввод-вывод без  
// преобразования символов конца строки)  
};

Конструкторы объектов (для классов ifstream,ofstream,fstream) и функции открытия/закрытия файлов:

ifstream();        // Без открытия файлов  
ifstream(          // С открытием файла в заданном   
char \*name,   // режиме imode   
int imode=ios::in,   
int prot=filebuf::openprot);

ifstream(int fd);  // С присоединенем файла с дескрип-  
// тором fd

ifstream(     // То же, с явно заданным буфером  
int fd,   
char \*buf, int sz);

void ifstream::open(      
char \*name,   // Открытие файла в заданном режиме   
int imode=ios::in,   
int prot=filebuf::openprot);

void close(); // Закрыть файл   
void setbuf(  
char \*p,int sz);// Установить буфер потока   
int  fd();         // Дескриптор открытого в потоке файла  
int  is\_rtl\_open();     // 1 - файл открыт в потоке

Унаследованные переопределения операторов позволяют проверять наличие ошибок в потоках в виде:

fstream  ss;

if (ss) ...   или  if (!ss) ...

Обработка исключительных ситуаций

Средства обработки ошибочных ситуаций позволяют передать обработку исключений из кода, в котором возникло исключение,   некоторому другому программному блоку, который выполнит в данном случае некоторые определенные действия. Таким образом, основная идея данного механизма состоит в том, что функция проекта, которая обнаружила непредвиденную ошибочную ситуацию, которую она не знает, как решить, генерирует сообщение об этом (бросок исключения). А система вызывает по этому сообщению программный модуль, который перехватит исключение  и отреагирует на возникшее нештатное событие. Такой программный модуль называют «обработчик» или перехватчик исключительных ситуаций. И в случае возникновения исключения  в его обработчик передаётся произвольное количество информации с  контролем ее типа. Эта информация и является характеристикой возникшей нештатной ситуации.

Обработка исключений в С++ это обработка с завершением. Это означает, что   исключается невозможность возобновления выполнения программы в точке возникновения исключения.

Для обеспечения работы такого механизма были введены следующие ключевые слова:

try   - проба испытания;

catch    - перехватить (обработать);

throw         - бросать.

Кратко рассмотрим их назначение.

**try** - открывает блок кода, в котором может произойти ошибка; это обычный составной оператор:

try { код};

Код содержит набор операций и операторов, который и будет контролироваться на возникновение ошибки. В него могут входить  вызовы  функции пользователя, которые компилятор так же возьмет на контроль. Среди данного набора операторов и операций обязательно указывают операцию броска исключения:  **throw**.

Операция броска **throw** имеет следующий формат:

throw  выражение;

где - «выражение» определяет тип информации, которая и описывает исключение (например, конкретные типы данных).

**catch** - сам обработчик исключения, который перехватывает информацию:

catch ( тип параметр){код  }

Через параметр обработчику передаются данные определенного типа, описывающие обрабатываемое исключение.

Код определяет те действия, которые надо выполнить при возникновении данной конкретной ситуации. В С++ используют несколько форм обработчиков. Такой обработчик получил название параметризованный специализированный перехватчик

Перехватчик должен следовать сразу же после блока контроля, т.е. между обработчиком и блоком контроля не должно быть ни одного оператора. При этом в одном блоке контроля можно вызывать исключения разных типов для разных ситуаций, поэтому обработчиков может быть несколько.

В этом случае их необходимо расположить сразу же за контролирующим блоком последовательно друг за другом.

Кроме того, запрещены переходы, как извне в обработчик, так и между обработчиками.

Можно воспользоваться универсальным или абсолютным обработчиком:

catch ( . . . ) {код }

где (...) - означают способность данного перехватчика обрабатывать информацию любого типа. Такой обработчик располагают последним в пакете специализированных обработчиков. Тогда, если исключение не будет перехвачено специализированными обработчиками, то буде выполнен последний - универсальный.

В случае не возникновения исключения,  набор обработчиков будет обойден, т.е. проигнорирован.

Если же исключение было брошено, при возникновении критической ситуации, то будет вызван конкретный перехватчик при совпадении его параметра с выражением в операторе броска, т.е. управление будет передано найденному обработчику. После выполнения кода вызванного обработчика, управление передается  оператору, который расположенный за последним перехватчиком, или проект корректно завершает работу.

Существенное отличие вызова конкретного обработчика от вызова обычной функции заключается в следующем: при возникновении исключения и передаче управления определенному обработчику, система осуществляет вызов всех деструкторов для всех объектов классов, которые были созданы с момента начала контроля  и до возникновения исключительной ситуации с целью их уничтожения.

Блоки **try**, как составные блоки могут быть вложены:

try { ...try {...  }...}

тогда, в случае возникновения исключения в некотором текущем блоке, поиск обработчика последовательно продолжается в блоках, предшествующих уровней вложенности с продолжением вызова деструкторов.

## Задание к работе

Общая постановка. Создать программу с абстрактным базовым классом и множественным наследованием, реализовать в нем:

* конструктор,
* деструктор,
* виртуальную функцию просмотра текущего состояния объекта print(),
* friend,
* функцию Run ().

Производные классы должны содержать переопределенную функцию просмотра состояния объектов (print()). Используя стандартные файловые потоки, информацию об объектах вывести в файл. При вводе - выводе данных обработка ошибочных ситуаций (например, невозможность открыть файл или нехватка места на диске) должно обрабатываться с использованием механизма исключений.

## Варианты заданий

1. Книги       
   (Название, Автор, жанр, год, страниц, тираж, отпечатано листов - функция Run ())