ЛАБОРАТОРНАЯ РАБОТА № 6
НАСЛЕДОВАНИЕ

Цель работы – изучить иерархии классов, механизмы работы с наследованием, научиться обеспечивать вызов методов и полей классов для обработки данных при множественном наследовании, изменять видимости компонент в определении класса с использованием спецификаторов доступа.

Теоретические сведения

Наследование классов

Наследование – это процесс создания новых классов на основе уже имеющихся. Имеющиеся классы обычно называются ***базовыми***, а новые классы, формируемые на основе базовых,– ***производными*** или ***наследниками***. Базовый класс определяет все те качества, которые будут общими для всех производных от него классов. В сущности, базовый класс представляет собой наиболее общее описание ряда характерных черт. Производный класс наследует эти общие черты и добавляет свойства, характерные только для него.

Наследуемые компоненты не перемещаются в производный класс, а остаются в базовых классах. Сообщение, обработку которого не могут выполнить методы производного класса, автоматически передается в базовый класс. Если для обработки сообщения нужны данные, отсутствующие в производном классе, то они автоматически и незаметно для программиста разыскиваются в базовом классе.

В определении и описании производного класса приводится список

базовых классов, из которых он непосредственно наследует данные и методы. Между именем вводимого нового класса и списком базовых классов помещается двоеточие.

Для того, чтобы показать, что класс B является наследником класса A, в определении класса B после имени класса ставится двоеточие и указывается имя базового класса:

```
class A
{
   public:
      A(); //конструктор класса A
      ~A(); //деструктор класса A
      MethodA(); //функция-член класса A
};
class B: public A //все компоненты класса A наследуются классом B
{
   public:
      B(); //собственный конструктор класса B
      ~B(); //собственный деструктор класса B
      ... //собственные методы класса B
};
```

При объявлении объекта класса B можно вызвать метод класса A:

```
B b; b.MethodA();
```

Особенно следует отметить, что конструктор базового класса всегда вызывается и выполняется до конструктора производного класса, а деструкторы – наоборот, то есть порядок уничтожения объекта противоположен порядку его конструирования.

При наследовании классов важную роль играет статус доступа (статус внешней видимости) компонентов. Для любого класса все его компоненты лежат в области его действия. Тем самым любая принадлежащая классу функция может использовать любые компонентные данные и вызывать любые принадлежащие классу функции. Вне класса в общем случае доступны только те его компоненты, которые имеют статус public.

Возможность использования объектами производного класса компонентов базового класса определяется правами доступа: наследник получает доступ только к компонентам, объявленным со спецификаторами *protected* и *public*.

Помимо простого наследования, существует еще и множественное наследование. Подробнее об особенностях наследования и правах доступа можно прочитать в [3], [4] и других учебниках по C++.

Виртуальные функции

В базовом и производном классах могут присутствовать методы с одинаковыми именами. В этом случае собственный метод производного класса «перекрывает» видимость метода базового класса, и при использовании объекта производного класса будет вызван метод именно этого класса. Однако, когда используется несколько объектов классов, производных от одного базового, в этом случае придется писать последовательность однотипных вызовов.

Для упрощения подобных ситуаций существует возможность обращаться к объектам производных классов через указатели на базовый класс. Чтобы для каждого объекта вызывался соответствующий метод, используется механизм **виртуальных функций**. Классы, включающие в себя виртуальные функции, называются ***полиморфными***, а сама возможность работать с группой разнородных объектов одинаковым образом, не задумываясь о различиях в реализации, называется **полиморфизмом**. (В сущности, для полиморфизма как такового виртуальные функции и не нужны. Зато они нужны для полиморфизма динамического (времени выполнения) в противовес статическому (времени компиляции), примерами которого являются перегрузка функций и использование шаблонов функций.)

Виртуальная функция – это функция класса, которая может быть переопределена в классах-наследниках так, что конкретная ее реализация для вызова будет определяться во время исполнения. Для объявления функции виртуальной в заголовок функции в базовом классе нужно добавить

служебное слово ***virtual***:

```
virtual тип_результата имя_функции (спецификация_параметров);
```

Для каждого класса, имеющего хотя бы один виртуальный метод, создаётся ***таблица виртуальных методов (функций)***. Каждый объект хранит указатель на таблицу своего класса. Для вызова виртуального метода используется такой механизм: из объекта берётся указатель на соответствующую таблицу виртуальных методов, а из нее – указатель на реализацию метода, используемого для данного класса. Такой подход называется ***поздним*** или ***динамическим связыванием***.

Базовый класс может и не предоставлять реализации виртуального метода, а только декларировать его существование. Такие методы без реализации называются ***чистыми виртуальными функциями*** и объявляются так:

```
virtual тип_результата имя_функции (спецификация_параметров) = 0;
```

В этой записи конструкция « = 0» называется *чистым спецификатором*. Такая функция ничего не делает и недоступна для вызовов. Ее назначение – служить основой для подменяющих ее функций в производных классах.

Класс, содержащий хотя бы одну чистую виртуальную функцию, называется ***абстрактным***, объект такого класса создать нельзя, но можно определять указатели и ссылки на абстрактные классы. Наследники абстрактного класса должны предоставить реализацию для всех его чистых виртуальных функций.

Статические члены класса

При создании объектов, с одной стороны, каждый объект имеет свои собственные независимые поля данных, с другой – все объекты одного класса используют одни и те же методы. Методы класса создаются и размещаются в памяти компьютера всего один раз – при создании класса, так как нет никакого смысла держать в памяти копии методов для каждого

объекта, поскольку у всех объектов методы одинаковые. А поскольку наборы значений полей у каждого объекта свои, поля объектов не должны быть общими. Однако существует ряд ситуаций, когда необходимо, чтобы все представители одного класса включали в себя какое-либо одинаковое значение. Для этих целей служат ***статические элементы класса***.

Статический элемент класса может рассматриваться как глобальная переменная или функция, доступная только в пределах области класса. Для определения статических полей и методов используется ключевое слово ***static***.

Статический элемент данных разделяется всеми представителями данного класса. То есть существует только один экземпляр переменной независимо от числа созданных представителей. Память под статический элемент выделяется, даже если не существует никаких представителей класса. Определение статических полей класса происходит не так, как для обычных полей. Обычные поля объявляются (компилятору сообщается имя и тип поля) и определяются (компилятор выделяет память для хранения поля) при помощи одного оператора. Для статических полей эти два действия выполняются двумя разными операторами: объявление поля находится внутри определения класса, а определение, как правило, располагается вне класса и зачастую представляет собой определение глобальной переменной. Например, подсчитаем число созданных представителей класса:

```
class A
{
private:
   static int iCount; // Объявление статического элемента данных
public:
   A() {iCount++;};
   int GetCount() {return iCount;};
};
int A::iCount = 0; //Определение статического элемента данных
int main()
{
```

```
  A a1, a2, a3; //создаем три объекта
  cout << "Число объектов " << a1.GetCount() << endl;
  cout << "Число объектов " << a2.GetCount() << endl;
  cout << "Число объектов " << a3.GetCount() << endl;
  return 0;
}
```

Поскольку конструктор вызывается трижды, то инкрементирование поля *iCount* также происходит трижды. Поэтому в результате выполнения программы будет напечатано:

```
Число объектов 3
Число объектов 3
Число объектов 3
```

Помимо статических полей, класс может иметь и статические функции. О том, что это такое и как с ними работать, можно прочитать в [3], [4] и других учебниках по C++.

Постановка задачи

1. Описать три класса: базовый класс «Строка» и производные от него класс «Строка-идентификатор» и класс, заданный индивидуальным вариантом. Обязательные для всех классов методы: конструктор без параметров, конструктор, принимающий в качестве параметра Си-строку, конструктор копирования, деструктор, перегрузка операции присваивания «=». Во всех методах всех классов предусмотреть печать сообщения, содержащего имя метода. Для конструкторов копирования каждого класса дополнительно предусмотреть диагностическую печать количества его вызовов, рекомендуется использовать статические члены класса.

Поля класса «Строка»: указатель на блок динамически выделенной памяти для размещения символов строки, длина строки в байтах. Обязательные методы, помимо вышеуказанных: конструктор, принимающий в качестве параметра символ (char), функция получения длины строки.

Строки класса «Строка-идентификатор» строятся по правилам записи идентификаторов в Си, и могут включать в себя только те символы, которые могут входить в состав Си-идентификаторов. Если исходные данные противоречат правилам записи идентификатора, то создается пустая «Строка-идентификатор».

Помимо обязательных компонентов классов, указанных в общей постановке задачи и в вариативной его части, при необходимости можно добавить дополнительные поля и методы.

2. Написать тестовую программу, которая должна:

- динамически выделить память под массив указателей на базовый класс (4-6 шт.);
- в режиме диалога заполнить этот массив указателями на производные классы, при этом экземпляры производных классов должны создаваться динамически с заданием начальных значений;
- для созданных экземпляров производных классов выполнить проверку всех разработанных методов с выводом исходных данных и результатов на дисплей.

Режим диалога должен обеспечиваться с помощью иерархического меню. Основные пункты:

1. «Инициализация». Подпункты:

1.1 «Число элементов». Задает число элементов в массиве указателей на базовый класс. После ввода числа элементов пользоваться этим пунктом меню запрещается.

1.2 «Начальное значение». С помощью этого пункта меню можно задать номер элемента, его тип и начальное значение. Задавать начальные значения и работать с другими пунктами меню запрещается до тех пор, пока

не будет задано число элементов. Допускается задать новое начальное значение несколько раз.

2. «Тестирование». Подпункты:
    2.1 «Строка»
    2.2 «Строка-идентификатор»
    2.3 Класс, соответствующий варианту задания
    2.4 «Задать операнды»

После выбора одного из этих пунктов меню предлагается выбрать один из методов из списка всех обязательных методов (кроме конструкторов и деструкторов), связанных с выбранным подпунктом.

3. Выход

Варианты заданий

Вариант 17

Дополнительные методы для класса «Строка-идентификатор»: перевод всех символов строки (кроме цифр) в нижний регистр, переопределение операции вычитания «-» (из первого операнда удаляются все символы, входящие во второй операнд).

Производный от «Строка» класс «Десятичная строка».

Строки данного класса могут содержать только символы десятичных цифр и символы «-» и «+», задающие знак числа, которые могут находиться только в первой позиции числа, при отсутствии знака число считается положительным. Если в составе инициализирующей строки будут встречены любые символы, отличные от допустимых, «Десятичная строка» принимает нулевое значение. Содержимое данных строк рассматривается как десятичное число.

Обязательные методы: определение, можно ли представить данное число в формате unsigned int, перегрузка операций вычитания «-» для получения разности двух десятичных чисел.