ЯЗЫКИ ПРОГРАММИРОВАНИЯ и СТРУКТУРЫ ДАННЫХ

Лекция 8

Объектно-ориентированное программирование.

Полиморфизм.

Наследование.

Виртуальные функции.

https://do.ssau.ru/moodle/course/view.php?id=1375

https://do.ssau.ru/moodle/mod/forum/view.php?id=34435

Полиморфизм

Основывается на возможности включения в данные объекта также и информации о методах их обработки.

Полиморфный метод обладает тем свойством, что при отсутствии полной информации о том, объект какого из классов в данный момент обрабатывается, он тем не менее корректно вызывается в том виде, который соответствует именно объекту этого класса.

Наследование

Порождённый класс наследует описание **основного класса. Порождённый класс** может изменяться с помощью добавления членов, изменения функций существующих членов и изменения привилегий доступа.

Удобство этого понятия можно показать на примере таксономической классификации, компактно подводящей итог больших областей знания.

- Например, если известно понятие "млекопитающее", а так же, что слон и мышь являются млекопитающими, то их описания можно сделать значительно более сжатыми, чем в другом случае.
- Корневое понятие "млекопитающее" содержит информацию о том, что млекопитающие — это теплокровные животные, относящиеся к высшим позвоночным, которые выкармливают своих детёнышей молоком, производящимся в их молочных железах.
- Эта информация унаследована и мышью, и слоном, но она выражается только один раз: в корневом понятии "млекопитающее".
- В C++ это означает, что признаки обоих классов, слона и мыши, порождены от основного класса, "млекопитающее".

Иерархия классов

- **Иерархия классов,** определённая общим наследованием, создаёт связанный набор типов пользователя, на которые можно ссылаться с помощью указателя базового класса.
- При обращении через этот указатель в C++ выбирается соответствующее функциональное определение во время выполнения.
- Каждый объект "знает", как на него должны воздействовать.
- Эта форма полиморфизма называется **чистым полиморфизмом**.
- С использованием наследования ключевыми элементами методологии ООП становятся:
  - разработка соответствующего набора типов;
  - проектирование их возможных связей и применение механизма наследования для совместного использования кода;
  - использование особых функций для полиморфической обработки связанных объектов.

Наследование

Новый, или производный **класс** может быть **определен на основе** уже имеющегося, или **базового**.

Новый класс сохраняет все свойства старого: данные объекта базового класса включаются в данные объекта производного, методы базового класса могут быть вызваны для объекта производного класса, причем они будут выполняться над данными включенного в него объекта базового класса.

- Для порождения нового класса от существующего используется следующая форма записи:

```
class имя класса:(public | protected | private)
имя базового класса
{
  объявления членов
};
```

Если ***уровень_ доступа*** не указан, то для производного класса по умолчанию используется спецификатор **`private`**, а для производной структуры - **`public`**

Защищённые члены класса

Квалификатор `protected`

- Защищённый член класса, также как и закрытый (`private`), недоступен вне класса.

Отличие:

- При открытом наследовании закрытые члены класса остаются закрытыми.
- Защищённые члены базового класса становятся защищёнными членами производного класса.
- Защищённые члены класса являются закрытыми, но могут наследоваться производным классом.
- Если производный класс является базовым по отношению к другому производному классу, то любой защищённый член исходного базового класса, открыто наследуемый первым производным классом, также может наследоваться вторым производным классом как защищённый член

Уровни доступа при наследовании

`public`

- Все открытые и защищённые члены базового класса становятся открытыми и защищёнными членами производного класса.
- Закрытые члены базового класса не меняют своего статуса и остаются недоступными членам производного.

`private`

- Все открытые и защищённые члены базового класса становятся закрытыми членами производного класса.
- Они остаются доступными членам производного класса, но недоступны остальным элементам программы, не являющимся членами базового или производного класса.

`protected`

- Все открытые и защищённые члены базового класса становятся защищёнными членами производного класса.

Видимость унаследованных членов

- Одна из особенностей порождённого класса — видимость унаследованных членов.
- Для определения доступности членов основного класса порождённому классу используются ключевые слова `public`, `protected` и `private`.

Пример порождения класса:

```
class student
{  protected:
    int     student_id;
    int     year;
    char    name[30] ;
   public:
    student(char* nm, int id, int y) ;
    void print();
};
class grad student: public student
{  protected:
    char     dept[10];
    char     thesis [80] ;
   public:
    grad_student (char* nm, int id, int y, char* d, char* th);
    void print();
};
```

Видимость унаследованных членов

- Одна из особенностей порождённого класса — видимость унаследованных членов.
- Для определения доступности членов основного класса порождённому классу используются ключевые слова `public`, `protected` и `private`.

Пример порождения класса:

```
class student
{  protected:
    int    student_id;
    int    year;
    char   name[30] ;
   public:
    student(char* nm, int id, int y) ;
    void print();
};
class grad student: public student
{  protected:
    char   dept[10];
    char   thesis [80] ;
   public:
    grad student (char* nm, int id, int y, char* d, char* th);
    void print();
};
```

В этом примере **`grad_student`** - порожденный класс, а **`student`** - базовый класс. Использование ключевого слова `public` с последующим двоеточием в заголовке порожденного класса означает, что *`protected`* и *`public`* члены класса **`student`** должны быть унаследованы как *`protected`* и *`public`* члены **`grad_student`** соответственно. Члены *`private`* **недоступны**.

Общее наследование также означает, что полученный класс **`grad_student`** – подтип класса **`student`**.

Открытое наследование

D открыто наследует классу B

```
class B
{
    ///
};
class D: public B
{
    ///
};
```

- Каждый объект типа ***D*** также является объектом типа ***B***, но *не наоборот.*
- ***B*** представляет собой более общую концепцию, чем ***D***, а ***D*** - более конкретную концепцию, чем ***B***.
- Везде, где может быть использован объект ***B***, может быть использован и объект ***D***, поскольку объект типа ***D*** является объектом типа ***B***.
- Если необходим объект типа ***D***, объект ***B*** не подойдет: каждый объект класса ***D*** «есть разновидность» ***B***, но не наоборот.

Открытое наследование – есть реализация отношения «является»

Открытое наследование

- Порождённый класс D представляет собой модификацию основного класса B, которая наследует общие и защищённые члены базового класса B.
- Такой механизм выгоден по следующим соображениям:
  - Код используется многократно. Тип D использует существующий проверенный код из B.
  - Различные полиморфные механизмы позволяют коду пользователя обрабатывать D как подтип **B**, что упрощает код пользователя, в тоже время предоставляя ему выгоды при обработке этих различий между подтипами.
- Часто порождённый класс добавляет новые члены к уже существующим членам класса
- Функции-члены (методы) также могут перекрывается. Это означает то, что порождённый класс имеет реализацию функции-члена, отличающуюся от базового класса.

!!! Не следует путать с перегрузкой, где то же самое имя функции может иметь различные значения для каждой уникальной сигнатуры.

Конструкторы и деструкторы при наследовании

- Конструкторы вызываются в иерархическом порядке, деструкторы – в обратном
  - При создании объекта производного класса сначала вызывается конструктор базового класса, а затем – производного
  - Конструкторы порождённого класса могут вызывать конструкторы базового класса. При этом используется такая же синтаксическая конструкция, как и для инициализации членов.

```
заголовок функции: имя базового класса (список параметров)
```

  - При уничтожении объекта производного класса сначала вызывается конструктор производного класса, а затем – базового

Виртуальные функции

- Ключевое слово **`virtual`** - функциональный спецификатор, обеспечивающий механизм выбора во время выполнения соответствующей функции-члена среди функций основных и порождённых классов.
- Функция порождённого класса автоматически становится **`virtual`**.
- Комбинация виртуальных функций и общего наследования представляет собой форму чистого полиморфизма и является наиболее общим и гибким способом формирования программного обеспечения.
- Специальные ограничения: деструкторы могут быть виртуальными, а конструкторы - нет.

Виртуальные функции

- Виртуальная функция является обычным выполняемым кодом. Семантика её вызова такая же, как и у других функций.
- В порождённом классе она может быть переопределена, и функциональный прототип порождённой функции должен иметь соответствие сигнатуры и типа возврата.
- Выбор того, какое определение функции вызвать для виртуальной функции, производится динамически.
- Типичный случай - когда основной класс имеет виртуальные функции, а порождённые классы имеют свои версии этих функций.
  - Указатель на базовый класс может указывать или на объект базового класса, или на объект порождённого класса.
  - Выбранная функция-член зависит от класса, на объект которого указывается, но не от типа указателя.
  - При отсутствии члена порождённого типа по умолчанию используется виртуальная функция основного класса.

Пример выбора виртуальной функции

```
class B {
    public:
         int i ;
    virtual void print_i()
                  { cout << i << " inside B\n"; }
};
class D: public B {
    public:
             void print_i()
           { cout << i << " inside D\n"; }
              //так же virtual
};

main()
{   B  bx; B* pbx = &bx; D dx ;
   dx.i = 1 + (bx.i = 1);
   pb -> print_i ();
   pb = & dx;
   pb -> print_i ();
}
Результат вывода этой программы:

1 inside B
2 inside D
```