Лекция 5

Методы и системы программирования. Основные принципы объектно-ориентированного программирования

Интерпретатор — это программа, которая получает исходную программу и по мере распознавания конструкций входного языка реализует действия, описываемые этими конструкциями.

Интерпретатор берет очередной оператор языка из текста программы, анализирует его структуру, проверяет на синтаксис и семантику, переводит в машинный код и затем сразу исполняет. Только после того, как текущий оператор успешно выполнен, интерпретатор перейдет к следующему. При этом, если один и тот же оператор должен выполняться в программе многократно, интерпретатор всякий раз будет выполнять его так, как будто встретил впервые.

Для выполнения такой программы на другом компьютере также должен быть установлен интерпретатор.

Транслятор — это программа, которая принимает исходную программу и порождает на своем выходе программу, записываемую на объектном языке программирования (объектную программу). В частном случае объектным может служит машинный язык, и в этом случае полученную на выходе транслятора программу можно сразу же выполнить на ЭВМ. В качестве объектного языка может служить и некоторый промежуточный язык, например, язык Ассемблера.

Для промежуточного языка может быть использован другой транслятор или интерпретатор — с промежуточного языка на машинный.

Трансляторы полностью обрабатывают весь текст программы, проверяя его на синтаксис, семантику, нередко при этом выполняя оптимизацию с помощью набора методов, позволяющих повысить быстродействие программы, затем переводят (транслируют) текст на машинный язык — генерируют машинный код. В результате законченная программа получается компактной и эффективной, работает в сотни раз быстрее программы, выполняемой с помощью интерпретатора, и может быть перенесена на другие компьютеры с процессором, поддерживающим соответствующий машинный код.

Транслятор с языка высокого уровня называют **компилятором**.

Разные типы процессоров имеют разные наборы команд.

Если язык программирования ориентирован на конкретный тип процессора и учитывает его особенности, то он называется языком программирования ***низкого уровня***.

чтение

Пример: языки Ассемблера.

Языки программирования ***высокого уровня*** значительно ближе и понятнее человеку, нежели компьютеру. Особенности конкретных компьютерных архитектур в них не учитываются, поэтому создаваемые программы на уровне исходных текстов легко переносимы на другие платформы, для которых создан транслятор этого языка.

Примеры: языки Basic, Pascal, C++, C# и т. д.

Fortran (Фортран) - Джим Бэкус в 50-е годы.

Cobol (Кобол) - начало 60-х годов.

чтение

Algol (Алгол) - 1960 год.

Pascal (Паскаль) - Никлаус Вирт в конце 70-х годов.

Basic (Бейсик) - в 60-х годах.

C (Си) - лаборатория Bell в 70-е годы, Деннис Ритчи.

C++ (Си++) - Бьярн Страуструп в 1980 году.

Java (Джава, Ява) - компания Sun в начале 90-х годов на основе Си++.

C# - 2000 г.

Языки программирования баз данных

чтение

СУБД (Системы Управления Базами Данных):
SQL Server (Microsoft), DB2 (IBM), Oracle, Adabas (Software AG), Informix и Sybase, dBase II (для ПК), FoxPro и Clipper, MySQL …

SQL (Structured Query Language) - структурированный язык запросов:

Oracle — PL/SQL,

Informix — INFORMIX 4GL,

Adabas — Natural и т. д.

Языки программирования для Интернета**:**

HTML, **Perl, Tcl/Tk**, **VRML**…

чтение

Языки моделирования: **CASE**-системы - формальные нотации **IDEF**, язык графического моделирования **UML**

--------- Другие языки программирования ----------

PL/I (ПЛ/1) - в 1964 году компания IBM - Programming Language One.

Smalltalk (Смолток) - в 1970 году в лаборатории корпорации XEROX.

LISP (Лисп) - в 1960 году Джоном Маккарти.

Prolog (Пролог) - в начале 70-х годов Аланом Колмероэ.

Ada (Ада) - в 1980 году.

Forth (Форт) - Чарльз Мур в 70-х годах.

Стили программирования

Одним из важнейших признаков классификации языков программирования является принадлежность их к одному из стилей, основными из которых являются следующие стили:

процедурный,
функциональный,
логический,
объектно-ориентированный.

Процедурное программирование

Процедурное (императивное) программирование является отражением архитектуры традиционных ЭВМ, которая была предложена фон Нейманом в 40-х годах. Теоретической моделью процедурного программирования служит алгоритмическая система под названием «машина Тьюринга».

чтение

Язык Макроассемблера.

Язык программирования С (Си) (UNIX в начале 70-х годов).

Basic(Бэйсик) (Beginners All-purpose Symbolic Instruction Code).

Pascal (Паскаль) – характеризуется: строгой типизированностью; высоким уровнем; широкими возможностями; стройностью, простотой и краткостью; строгостью, способствующей написанию эффективных и надежных программ; высокой эффективностью реализации на ЭВМ (Никлаус Вирт).

Процедурное программирование

чтение

Особенности:

- необходимость явного управления памятью, в частности, описание переменных;
- малая пригодность для символьных вычислении;
- отсутствие строгой математической основы;
- высокая эффективность реализации на традиционных ЭВМ.

Концепция памяти как хранилища значений, содержимое которого может обновляться операторами программы, *является фундаментальной* в императивном программировании.

Концепция памяти

Все данные (входные, выходные и промежуточные) должны сохраняться в оперативной памяти.

- Для сохранения данных в памяти используется понятие **переменной.**
- **Переменная -** имя области памяти, в которой размещается входное, выходное или промежуточное значение (переменная содержит адрес первой ячейки памяти, выделяемой под хранение значения).
- Сначала в программе нужно **объявить переменную -** указать ее **имя** и определить ее **тип данных**.

Концепция памяти

чтение

Имя переменной - идентификатор, формируемый по определенным правилам (в языке C++):

1. Для образования идентификаторов могут быть использованы строчные или прописные буквы латинского алфавита, символ «подчеркивание» (_) и арабские цифры.
2. Не следует использовать цифры и символ «подчеркивание» (_) в качестве первого символа идентификатора.
3. Идентификатор не должен совпадать с ключевыми словами, с зарезервированными словами и именами функций библиотеки компилятора языка C/C++.
4. Допускается любое количество символов в идентификаторе, хотя некоторые компиляторы и компоновщики налагают на длину идентификатора ограничение, например, в C обычно значимыми являются первые 31 символ.
5. Язык регистрозависимый.

Например: abc, ABC, A128_B, a128b .

Концепция памяти

Имя переменной - идентификатор, формируемый по определенным правилам (в языке C#):

1. Для образования идентификаторов могут быть использованы строчные или прописные буквы латинского алфавита, арабские цифры, символ «подчеркивание» (_) и символ @.
2. Нельзя использовать цифры в качестве первого символа идентификатора.
3. Не следует использовать символ «подчеркивание» (_) в качестве первого символа идентификатора.
4. Символ @ можно использовать только в качестве первого символа идентификатора.

Концепция памяти

5. Идентификатор не должен совпадать с ключевыми словами, с зарезервированными словами и именами функций библиотеки компилятора языка C#.
6. Длина идентификатора не ограничена. Пробелы внутри имен не допускаются.
7. Язык регистрозависимый.
8. В идентификаторах C# разрешается использовать, помимо латинских букв, буквы национальных алфавитов. Например, правильными являются идентификаторы Фёкла и calc. Более того, можно применять даже так называемые *escape-последовательности Unicode*, то есть представлять символ с помощью его кода в шестнадцатеричном виде с префиксом \u, например, \u00F2.

Концепция памяти

Тип данных определяет:

- размер оперативной памяти, выделяемой под величину;
- внутреннее представление данных в памяти компьютера;
- множество (диапазон) значений, которые могут принимать величины этого типа;
- операции и функции, которые можно применять к величинам этого типа.

Пример объявления переменных:

int abc; float ABC; char a128b.

Концепция памяти

Основным является **оператор присваивания**, служащий для изменения содержимого областей памяти.

abc = 5; ABC = 6.39; a128b = '+'.

Выполнение программы сводится к последовательному выполнению операторов с целью преобразования исходного состояния памяти, то есть значений исходных данных, в заключительное, то есть в результаты.

Функциональное программирование

Выражение: скалярные константы, структурированные объекты, функции, тела функций и вызовы функций.

Функция – это однозначное отображение из $X_п$ в X, где X — множество выражений.

чтение

Аппликативный язык программирования включает элементы:

- классы констант, которыми могут манипулировать функции;
- набор базовых функций, которые программист может использовать без предварительного объявления и описания;
- правила построения новых функций из базовых;
- правила формирования выражений на основе вызовов функций.

LISP(Лисп) (LISt Processing — обработка списков), 1959 г.

Функциональное программирование

Программа представляет собой совокупность описаний функций и выражения, которые необходимо вычислить. Данное выражение вычисляется посредством редукции, то есть серии упрощений, до тех пор, пока это возможно по следующим правилам: вызовы базовых функций заменяются соответствующими значениями; вызовы небазовых функций заменяются их телами, в которых параметры замещены аргументами.

чтение

Функциональное программирование не использует концепцию памяти как хранилища значений переменных. Операторы присваивания отсутствуют, вследствие чего переменные обозначают не области памяти, а объекты программы, что полностью соответствует понятию переменной в математике.

Функциональное программирование

В принципе, можно составлять программы и вообще без переменных. Кроме того, нет существенных различий между константами и функциями, то есть между программами и данными.

чтение

В результате этого функция может быть значением вызова другой функции и может быть элементом структурированного объекта.

Число аргументов при вызове функции не обязательно должно совпадать с числом параметров, указанных при ее описании.

Перечисленные свойства характеризуют аппликативные языки как языки программирования очень высокого уровня.

Логическое программирование

PROLOG (Пролог) (PROgramming in LOGic — программирование в терминах логики) - А. Кольмероэ в 1973 году.

чтение

Языки логического программирования характеризуются:

- высоким уровнем;
- строгой ориентацией на символьные вычисления;
- возможностью инверсных вычислений, то есть переменные в процедурах не делятся на входные и выходные;
- возможной логической неполнотой, поскольку зачастую невозможно выразить в программе определенные логические соотношения, а также невозможно получить из программы все выводы правильные.

Логическое программирование

Языки логического программирования, в особенности Пролог, широко используются в системах искусственного интеллекта.

чтение

Центральным понятием в логическом программировании является *отношение*. Программа представляет собой совокупность определений отношений между объектами (в терминах условий или ограничений) и цели (запроса). Процесс выполнения программы трактуется как процесс общезначимости логической формулы, построенной из программы по правилам, установленным семантикой используемого языка.

Логическое программирование

Результат вычисления является побочным продуктом этого процесса. В реляционном программировании нужно только специфицировать факты, на которых алгоритм основывается, а не определять последовательность шагов, которые требуется выполнить. Это свидетельствует о декларативности языка логического программирования. Она метко выражена в формуле Р. Ковальского:

чтение

«алгоритм = логика + управление».

Логические программы, в принципе, имеют небольшое быстродействие, так как вычисления осуществляются методом проб и ошибок, поиском с возвратами к предыдущим шагам.

Объектно-ориентированное программирование

чтение

SIMULA-67, SMALLTALK (А. Кей в 1972 году).

К наиболее современным объектно-ориентированным языкам программирования относятся C#, **C++**, Java и Object Pascal.

Язык **C++** начало 80-х годов Б. Страуструп, сотрудник лаборатории Bell корпорации AT&T.

В 1990 году сотрудник корпорации Sun Д. Гослинг на основе расширения **C++** разработал объектно-ориентированный язык Oak, основным достоинством которого было обеспечение сетевого взаимодействия различных по типу устройств. Новая интегрируемая в Internet версия языка, получила название Java.

Фирма Borland (Филипп Канн) — среда Delphi. Язык Pascal в Delphi был существенно дополнен по сравнению с версией Turbo Pascal и стал называться языком Object Pascal, а в 2002 году языком Delphi – разработчик Андерс Хейльсберг.

В 2000 году корпорация Microsoft анонсировала новый язык **C#** (один из авторов **Андерс Хейльсберг).**

Объектно-ориентированное программирование

Инкапсуляция, Наследование, Полиморфизм

В основе объектно-ориентированного стиля программирования лежит понятие объекта, а суть его выражается формулой:

«объект = данные + процедуры».

Каждый объект интегрирует в себе некоторую структуру данных и доступные только ему процедуры обработки этих данных, называемые методами. Объединение данных и процедур в одном объекте называется ***инкапсуляцией*** и присуще объектно-ориентированному программированию.

Объектно-ориентированное программирование

Инкапсуляция, Наследование, Полиморфизм

Для описания объектов служат **классы**.

Класс – тип данных - определяет свойства и методы объекта, принадлежащего этому классу.

Соответственно, любой объект можно определить как экземпляр класса (переменная типа класс).

Программирование рассматриваемого стиля заключается в выборе имеющихся или создании новых объектов и организации взаимодействия между ними.

При создании новых объектов свойства объектов могут добавляться или ***наследоваться*** от объектов-предков.

В процессе работы с объектами допускается ***полиморфизм*** — возможность использования методов с одинаковыми именами для обработки данных разных типов.

Структура класса

```
class имя_класса
{    // Объявление переменных экземпляра
     спецификатор доступа тип переменная1;
     спецификатор доступа тип переменная2;
   //...
     спецификатор доступа тип переменнаяN;
   // Объявление методов
    спецификатор доступа возращаемый_тип метод1 (параметры)
    {
       // тело метода
    }
   спецификатор доступа возращаемый_тип метод2 (параметры).
    {
        // тело метода
    }
   //...
   спецификатор доступа возращаемый_тип методN (параметры)
   {
       // тело метода
   }
}
```

Структура класса

спецификатор (модификатор) доступа:

- **public** (открытый);
- **private** (закрытый - видимый в пределах класса - **по умолчанию**);
- **protected** (защищенный - видимый в пределах класса и производных классов - наследников);
- **internal** (внутренний - служит в основном для сборки, которая в широком смысле означает в C# разворачиваемую программу или библиотеку).

метод1 - методN - имена методов (функций (подпрограмм), размещенных в классе). Методы содержат код для обработки данных (полей) класса. Метод может иметь имя **Main**() лишь в том случае, если программа начинается с данного класса.

Структура метода класса

спецификатор доступа
возращаемый_тип **имя_метода** (параметры)
{
// тело метода
}

```
public void Area(int Area){// метод не возвращающий значение
  Console.WriteLine(" " + Area + " – общая площадь");
}
public int AreaP(int Area) {// метод возвращающий значение
  return Area = Area * Area;
}
```

Структура метода класса

Если возвращаемый_тип метода - **void** , то в теле метода оператор **return** может присутствовать для досрочного завершения работы метода. В этом случай он не должен возвращать значение: **return;**

Если возвращаемый_тип метода - **не void** , то в теле метода должен присутствовать хотя бы один оператор **return,** который обязательно возвращает значение типа возвращаемый_тип.

```
doubleA(double pi, double r) {
    return (2*pi*r*r);
}
```

Пример 1

```
class Building {
    public int Floors; // количество этажей
    public int Area; // общая площадь здания
    public int Occupants; // количество жильцов
}

// Объявление переменной house – экземпляра класса, которая является ссылкой на объект типа Building
Building house = new Building();
```

Пример 1

Всякий раз, когда объявляется экземпляр класса, создается также объект, содержащий собственную копию каждой переменной экземпляра, определенной в данном классе. Таким образом, каждый объект типа **Building** будет содержать свои копии переменных экземпляра **Floors**, **Area** и **Occupants**.

Для доступа к этим переменным служит оператор доступа к члену класса, который принято называть оператором-точкой: **объект.член**

Например, присваивание значения 2 переменной **Floors** объекта **house** осуществляется с помощью следующего оператора: house.Floors = 2;

В целом, оператор-точка служит для доступа к переменным экземпляра (**полям**) и **методам**.

Пример 2

```
using System;
class Building {    // Пользовательский класс
    public int Floors; // количество этажей
    public int Area; // общая площадь здания
    public int Occupants; // количество жильцов
    // Метод - выводит площадь на одного человека
    public void AreaPerPerson() {
        Console.WriteLine(" " + Area / Occupants + "
   приходится на одного человека");
    }
}
```

Пример 2

```
// Использовать метод AreaPerPerson()
class BuildingDemo {
    static void Main() {
        Building house = new Building();
        Building office = new Building();
        // Присвоить значения полям в объекте house
        house.Occupants = 4;
        house.Area = 2500;
        house.Floors = 2;
        // Присвоить значения полям в объекте office
        office.Occupants = 25;
        office.Area = 4200;
        office.Floors = 3;
```

Пример 2

```
        Console.WriteLine("Дом имеет:\n " + house.Floors
    + " этажа\ n " + house.Occupants + " жильца\ n " +
house.Area + "кв. футов общей площади, из них");
        house.AreaPerPerson();  // вызов метода
        Console.WriteLine ();
        Console.WriteLine("Учреждение имеет:\n " +
    office.Floors + " этажа\ n " + office.Occupants + "
    работников\ n " + office.Area + " кв. футов общей
    площади, из них");
        office.AreaPerPerson(); // вызов метода
     }
}
```

Интегрированные системы программирования включают:

1 Текстовый редактор

2 Программа-компилятор (.obj)

3 Библиотеки функций (.LIB)

4 Редактор связей или сборщик .EXE или .COM.

5 **Отладчик**

Среды быстрого проектирования

RAD-среды (Rapid Application Development)

чтение

- Basic: Microsoft Visual Basic
- Pascal: Borland Delphi
- C++: Borland C++Bulider
- Java: Symantec Café
- RAD Studio
- Visual Studio .NET

Контрольные вопросы !!!

1 Что такое язык программирования?

2 В чем различие компиляторов и интерпретаторов?

3 Объясните термины «язык низкого уровня» и «язык высокого уровня».

4 Какие языки программирования активно используются сегодня?

5 Охарактеризуйте основные стили программирования.

6 В чем состоит различие между естественными языками и языками программирования?

7 Что нужно для создания программы?

8 Что такое среды быстрого проектирования?