Лекция 6. Объектно-ориентированное программирование. Язык C++.

1. Введение в ООП.

1.1. Эволюция языков программирования.

Рассмотрим кратко эволюцию языков программирования.

1.1.1. Языки программирования низкого уровня.

Язык программирования, ориентированный на конкретный тип процессора, и, операторы которого близки к машинному коду, называется языком программирования низкого уровня. Термин "низкий уровень" обозначает его неразрывную связь с аппаратной частью компьютера.

Машинные коды. Первые ЭВМ появились в 1940-х годах и программировались с помощью машинных кодов. Машинный код состоял из последовательностей нулей и единиц. Каждая элементарная операция имела свой код, необходимо было явно указывать адреса ячеек памяти, в которых хранились данные, или куда их необходимо было сохранять. Такой подход содержал ряд неудобств, основными среди которых являлись:

- программа была машинно-зависимой, т. к. различные типы процессоров отличались друг от друга архитектурой и системой команд;
- чтение программы, а также изменение, отладка и поиск ошибок в ней вызывали огромные трудности.

Ассемблер. В начале 1950-х годов была осуществлена идея использования символьных имен вместо адресов данных и замены цифровых кодов операций на мнемонические (словесные) обозначения. Язык программирования, реализующий данный подход, получил название Ассемблер (от англ. assembler - сборщик). Программа, записанная на Ассемблере, не может обрабатываться непосредственно процессором. Возникла необходимость преобразования текста программы, записанной на данном языке, в машинный код. Для решения этой задачи были созданы трансляторы.

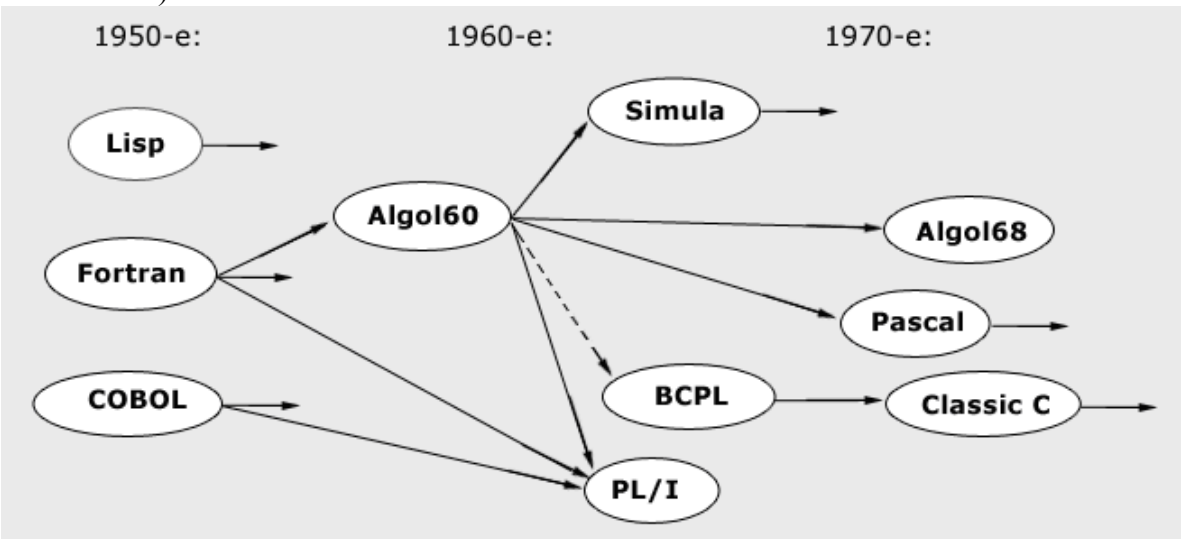
Транслятор – это программа, преобразующая текст, представленный на одном из языков программирования, в эквивалентную программу на машинном языке.

Использование символьных имен вместо числовых кодов операций и адресов данных существенно упростило процесс программирования, однако, по-прежнему сохранялась зависимость программы от типа процессора.

Необходимо отметить, что во многих сегодняшних задачах, где качество машинного кода ставится выше удобства программирования (например, при разработке ядра операционной системы), по-прежнему используется Ассемблер.

1.1.2. Языки программирования высокого уровня

Дальнейшая эволюция языков программирования привела к появлению языков высокого уровня, что позволило отвлечься от системы команд конкретного типа процессора. Ниже приведена диаграмма важнейших первых языков (источник: Б. Страуструп <Программирование: принципы и практика использования C++>)



Важное значение для развития высокоуровневых языков программирования имела разработка во второй половине 1950-х годов трех языков - Fortran, COBOL, Lisp. Философия, стоящая за этими языками, заключается в создании высокоуровневой системы обозначений, облегчающей программисту написание программ.

Язык программирования Fortran. Fortran (Formula Translation - трансляция формул) – был разработан в 1956 году сотрудниками компании IBM под руководством Джона Бэкуса. Появление Fortran – одно из наиболее значительных событий в истории языков программирования. Fortran предназначался для научных вычислений. Он обладал достаточно машинно-независимой системой команд: код на языке Fortran можно было переносить из одного компьютера в другой с минимальными изменениями, что явилось огромным достижением для своего времени. По этим причинам Fortran считается первым высокоуровневым языком программирования.

Язык программирования COBOL. COBOL (Common Business-Oriented Language – язык программирования для коммерческих и деловых задач) был разработан комитетом CODASYL в 1959-60 годах по инициативе Министерства обороны США и группы основных производителей компьютеров для обработки бизнес-данных. Основной упор в этом языке был сделан на манипуляции с данными: копирование, хранение и поиск, вывод на печать (отчеты).

Язык программирования Lisp. Lisp (LISt Processing – обработка списков) был разработан в 1958 году Джоном Маккарти для обработки связанных списков и символьной информации. Lisp был первым языком, не зависевшим от аппаратного обеспечения. В настоящее время существует множество диалектов языка. Это семейство языков нашло применение в области моделирования искусственного интеллекта.

Язык программирования Алгол (англ. Algol от algorithmic <алгоритмический> + англ. language <язык>) – название ряда языков программирования, применяемых при составлении программ для решения научно-технических задач на ЭВМ. Разработан комитетом по языку высокого уровня IFIP в 1958-1960 годах (Алгол 58, Алгол 60). Кардинально переработан в 1964-1968 годах (Алгол 68). Один из первых языков высокого уровня. Был популярен в Европе, в том числе в СССР, в качестве как языка практического программирования, так и академического языка (языка публикации алгоритмов в научных работах), но в США и Канаде не смог превзойти распространённый там Фортран. Оказал заметное влияние на все разработанные позднее императивные языки программирования. Обычно названием Алгол (без уточнения версии) именуют Алгол 60, в то время как Алгол 68 рассматривается как самостоятельный язык.

Язык программирования Паскаль (Pascal). Создан выдающимся специалистом в области computer science Никлаусом Виртом. Свое название язык получил в честь знаменитого французского ученого средневековья Блеза Паскаля. Впервые предварительное описание языка программирования Паскаль было опубликовано в 1968 году. Паскаль продолжал линию АЛГОЛ-W (язык, над которым Н. Вирт работал в 1965 году в рамках конкурса международной федерации по обработке информации (IFIP) на разработку нового языка программирования – преемника АЛГОЛ-60). Первый вариант компилятора появился в 1971 году, а спустя год вышли в свет соответствующие публикации. Паскаль стал одним из первых языков программирования, обладающих средствами структурного (процедурного) программирования, что стало его несомненным достоинством. Быстро растущий интерес к языку и небольшие его изменения привели к публикации в 1973 году так называемого Пересмотренного сообщения, в котором Паскаль (Pascal) определялся в терминах стандартов ISO.

Разрабатывая язык Паскаль, Никлаус Вирт изначально преследовал следующие основные цели:

- язык должен быть пригоден для обучения программированию;
- реализация языка должна быть эффективной и надежной на существующих ЭВМ.

Тем не менее, прозрачная структура, ясный синтаксис и большие возможности сделали язык популярным не только в образовательной среде, но и среди профессиональных программистов.

1.2. Различные подходы к программированию

1.2.1. Процедурный подход.

На заре становления программирования (написания программ для ЭВМ) программы имели простейшую структуру. Первый подход к их созданию состоял в разработке отдельных подпрограмм (модулей, процедур), которые можно было сохранять отдельно и использовать в других программах повторно. Поэтому первый поход к программированию назвали ***процедурным***.

1.2.2. Структурный подход.

Вторым этапом развития программирования можно назвать ***структурный подход*** к программированию. Прежде всего это разбиение сложных структур данных на элементарные части и программирование обработки базовых типов данных. Также этот подход подразумевает и обратное объединение в структуру различных типов данных. В его рамках возникли разные принципы

декомпозиции сложных систем с целью их реализации в виде небольших подпрограмм. На основе структурного подхода возникли такие известные языки программирования, как C, Pascal, ALGOL.

1.2.3. Объектно-ориентированный подход.

На третьем этапе развития программирования появился так называемый ***объектный*** подход. ООП – объектно-ориентированное программирование – технология создания сложного программного обеспечения, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого типа (или класса), а классы, в свою очередь, образуют целую иерархию с наследованием различных свойств. На основе ООП возникли такие языки как C++ , Java, C# и др.. Основными достоинствами ООП по сравнению с его предками является более естественная декомпозиция программного обеспечения, локализация данных, что позволяет вести практически полностью независимую разработку отдельных частей программы (объектов).

1.2.4. Современный подход

Современный четвёртый этап связан с введением CASE-технологий и компонентного подхода к разработке ПО. Новый вид программирования – компонентно-ориентированное программирование (КОП). Отличительной особенностью современного этапа развития технологии программирования является создание и внедрение автоматизированных технологий разработки и сопровождения программного обеспечения, которые были названы CASE-технологиями.

1.3 Объектно-ориентированное программирование (ООП), использующееся в C++.

1.3.1. Введение в ООП.

ООП – методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования. Идеологически ООП – подход к программированию как к моделированию информационных объектов, решающий на новом уровне основную задачу структурного программирования: структурирование информации с точки зрения управляемости, что существенно улучшает управляемость самим процессом моделирования, что, в свою очередь, особенно важно при реализации крупных проектов.

Управляемость для иерархических систем предполагает минимизацию избыточности данных (аналогичную нормализации) и их целостность, поэтому созданное удобно управляемым будет и удобно пониматься. Таким образом, через тактическую задачу управляемости решается стратегическая задача – трансляция понимания задачи программистом в наиболее удобную для дальнейшего использования форму.

Основные принципы структурирования в случае применения ООП связаны с различными аспектами базового понимания предметной задачи, которое требуется для оптимального управления соответствующей моделью:

- ***абстракция*** – выделение в моделируемом предмете важного для решения конкретной задачи свойства, формализуемое в виде класса;
- ***инкапсуляция*** – быстрая и безопасная организация иерархической управляемости, т. е. реализация простой команды "что делать", без одновременного уточнения "как именно делать" (последняя принадлежит другому уровню управления);
- ***наследование*** – быстрая и безопасная организация и использование родственных понятий (свойств, данных), которая применяется в целях учета на каждом иерархическом шаге только тех изменений, которые необходимы "здесь и сейчас", без дублирования остальных изменений, учтённых на предыдущих шагах иерархии;
- ***полиморфизм*** – механизм определения точки кода, в которой единое управление лучше распараллелить или наоборот – собрать воедино.

То есть фактически речь идёт о прогрессирующей организации информации согласно первичным семантическим критериям: <важное/неважное>, <ключевое/подробности>, <родительское/дочернее>, <единое/множественное>. ***Прогрессирование***, в частности, на последнем этапе даёт возможность перехода на следующий уровень детализации, что замыкает общий процесс.

Обычный человеческий язык в целом отражает идеологию ООП, начиная с инкапсуляции представления о предмете в виде его имени и заканчивая полиморфизмом использования слова в переносном смысле, что в итоге развивает выражение представления через имя предмета до полноценного понятия-класса.

1.3.2. Более точные определения

Абстракция данных – выделение значимой информации и исключение из рассмотрения незначимой. В ООП рассматривают лишь абстракцию данных (нередко называя её просто <абстракцией>), подразумевая набор наиболее значимых характеристик объекта, доступных остальной программе.

Инкапсуляция – свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе. Одни языки (например, C++, Java или Ruby) отождествляют инкапсуляцию с сокрытием, но другие (Smalltalk, Eiffel, OCaml) различают эти понятия.

Наследование – свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс – потомком, наследником, дочерним или производным классом.

Полиморфизм подтипов (в ООП называемый просто "полиморфизмом") – свойство системы, позволяющее использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта. Другой вид полиморфизма – параметрический – в ООП называют обобщённым программированием.

Класс – универсальный, комплексный тип данных, состоящий из тематически единого набора "полей" (переменных более элементарных типов) и "методов" (функций для работы с этими полями). Класс является моделью информационной сущности с внутренним и внешним интерфейсами для оперирования своим содержимым (значениями полей).

В классах широко используются специальные блоки из одного или чаще двух спаренных методов, отвечающих за элементарные операции с определённым полем (интерфейс присваивания и считывания значения), которые имитируют непосредственный доступ к полю. Эти блоки называются "свойствами" и почти совпадают по конкретному имени со своим полем (например, имя поля может начинаться со строчной, а имя свойства - с заглавной буквы). Другим проявлением интерфейсной природы класса является то, что при копировании соответствующей переменной через присваивание копируется только интерфейс, но не сами данные, то есть класс – ссылочный тип данных.

Переменная-объект, относящаяся к заданному классом типу, называется экземпляром этого класса. При этом в некоторых исполняющих системах класс также может представляться некоторым объектом при выполнении программы посредством динамической идентификации типа данных. Обычно классы разрабатывают таким образом, чтобы обеспечить отвечающие природе объекта и решаемой задаче целостность данных объекта, а также удобный и простой интерфейс. В свою очередь, целостность предметной области объектов и их интерфейсов, а также удобство их проектирования, обеспечивается наследованием.

Объект – сущность в адресном пространстве вычислительной системы, появляющаяся при создании экземпляра класса (например, после запуска результатов компиляции и связывания исходного кода на выполнение).

Класс описывает абстрактное поведение. Объектные типы строятся на основе класса посредством добавления различных приватных полей и методов. Объект (то есть значение объектного типа, традиционно называемое «экземпляром класса») порождается конструктором на основе начальных параметров.

2. Язык C++.

2.1. Причины возникновения.

Причиной возникновения языков, основанных на ООП – усложнение задач программирования при реализации сложных информационных систем и технологий. C++ – одна из первых таких реализаций.

2.2. Совместимость с языком C.

C++ включает себя большую часть элементов языка C. Изначально C++ – объектно-ориентированное расширение языка C.

2.3. Новые конструкции.

Новые конструкции в C++:

- комментарии;
- встраиваемые функции (оператор inline);
- задание аргументов по умолчанию;
- функции без аргументов и с неизвестными аргументами;
- описание переменных в середине блока;
- новый механизм передачи параметров;

- понятие ссылки;
- использование констант;
- ссылки на правосторонние значения;
- логический тип;
- модификация перечислимых типов с помощью классов;
- инициализация статических массивов;
- многомерные массивы;
- новые операторы управления динамической памятью new и delete;
- динамические массивы;
- классы и объекты;
- перегрузка функций.

Все отличия можно посмотреть здесь:
https://ru.wikibooks.org/wiki/%D0%A1%D0%B8%2B%2B/%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%8B%D0%B5_%D0%BE%D1%82%D0%BB%D0%B8%D1%87%D0%B8%D1%8F_%D0%A1%D0%B8%2B%2B_%D0%BE%D1%82_%D0%A1%D0%B8

3. Инкапсуляция.

3.1. Определения класса, структуры и объекта. Статические и константные члены класса. Спецификатор inline. Указатель this и указатели на члены класса. Область видимости и управление доступом к членам класса. Дружественные классы и методы.

3.2. Конструкторы и деструкторы. Конструктор по умолчанию, конструктор копирования и оператор присваивания. Спецификатор explicit. Автоматически создаваемые конструкторы и деструкторы. Порядок вызовов конструкторов и деструкторов.

4. Примеры

4.1 Оформление программы

4.2 Компиляция и выполнение