Появление машинного языка

Но с развитием компьютерной техники появился машинный язык, с помощью которого программист мог задавать команды, оперируя с ячейками памяти, полностью используя возможности машины. Суть этого языка — набор кодов, обязательно понятных процессору, к кому обращаются. Части («слова») этого языка называются *инструкциями*, каждая из которых представляет собой одно элементарное действие для центрального процессора, как, например, считывание информации из ячейки памяти. Если знаешь, можно непосредственно управлять процессором. Тогда ещё компьютеры были простыми вычислительными машинами, основная задача была считать. Но они развивались, а, понятно дело, использование большинства компьютеров на уровне машинного языка затруднительно, особенно это касается ввода-вывода. Поэтому со временем от его использования пришлось отказаться.

Например, для организации чтения блока данных с гибкого диска программист может использовать 16 различных команд, каждая из которых требует 13 параметров, таких как номер блока на диске, номер сектора на дорожке и т. п. Когда выполнение операции с диском завершается, контроллер возвращает 23 значения, отражающие наличие и типы ошибок, которые надо анализировать. Уже одно обращение к процессору громоздко, а уж анализ ошибок и вовсе представляется невообразимым, особенно, если не именно с этим процессором приходится работать. Вообще набор команд машинного языка сильно зависит от типа процессора(см.).

Язык ассемблера

На протяжении 1950-х годов запросы на разработку программного обеспечения возросли и программы стали очень большими. Приходилось писать очень много кода , хотя обеспечение и было весьма простым: по тем временам дизайн рабочего стола был проще нынешнего, программы работали с элементарными вещами, а компьютер только ещё начинал победно шествовать. Однако программы запутывались всё больше, их структура усложнилась, потому что всё время развивалась компьютерная техника. Тогда стали пользоваться специальными программами-сборщиками программ из маленьких кусочков кодов — ассемблерами. Пошёл новый этап развития.

Теперь, когда была нужна эффективная программа, вместо машинных языков использовались близкие к ним машинно-ориентированные языки ассемблера. К таковым относились, например, Autocode, с 1954-ого г. — IPL (предшественник языка LISP) и, с 1955-ого г. — FLOW-MATIC (предшественник языка COBOL). Теперь люди стали использовать мнемонические команды взамен машинных команд.

Но даже работа с ассемблером достаточно сложна и требует специальной подготовки. Например, для процессора Zilog Z80 машинная команда `00000101` предписывает процессору уменьшить на единицу свой регистр `B`. На языке ассемблера это же будет записано как `DEC B`.

Языки высокого уровня

Следующий шаг был сделан в 1954 году, когда был создан первый язык высокого уровня — Фортран (англ. *FORTRAN - FORmulaTRANslator*), а за ним и некоторые другие, как LISP, ALGOL 58, FACT (ещё один предшественник языка COBOL). Языки высокого уровня имитируют естественные языки, используя некоторые слова разговорного языка и

общепринятые математические символы. Эти языки более удобны для человека, с помощью них, можно писать программы до нескольких тысяч строк длиной. Конечно, это достижение было очень ценно. Условными словами можно было, как привычно человеку, гораздо более просто выразить сложную программную операцию из битов. Однако, легко понимаем в коротких программах, этот язык становился нечитаемым и трудно управляемым, когда дело касалось больших программ. То есть, простоты по-прежнему не хватало. Решение этой проблемы пришло после изобретения языков структурного программирования (англ. *structuredprogramminglanguage*), таких как Алгол(1958), Паскаль(1970), Си(1972). Но по порядку. 1959-й год — изобретён COBOL. 1962-й год — Симула. С него началась эпоха *структурного программирования*.

Появление структурного программирования

К тому времени люди начали понимать, что создание программного обеспечения — гораздо более сложная задача, чем они себе представляли. Это привело к разработке структурного программирования. С развитием структурного программирования следующим достижением были процедуры и функции. То есть, если есть задача, которая выполняется несколько раз, то её можно объявить как функцию или как процедуру и в выполнении программы просто вызывать её. Общий код программы в данном случае становится меньше. Это способствовало созданию модульных программ.

А следующим достижением было использование *структур*, благодаря которым перешли к классам. Структуры — это составные типы данных, построенные с использованием других типов. Например, структура времени: в неё входят: часы, минуты, секунды. Свою очередь и часы, и минуты, и секунды — они описаны при помощи других, более простых и более элементарных типов. Вместо сложной работы надо множеством типов, из которых каждый может быть со своими ограничениями — и что подходит одному типу, запрещено в другом, — вместо того программист бы мог создать структуру «время» и работать с ней, как с единым типом, где нету исключений и один формат.

Свою очередь, Класс — это структура, у которой свои переменные и функции, которые работают с этими переменными. То есть, это может быть названо особою средою, отличною ото других классов. Все классовые члены одного типу. См. подробнее статью.

Коротко, это достижение в области программирования было очень велико. Теперь программирование можно было разбить на классы и тестировать не всю программу, состоящую из 10 000 строк кода, а разбить программу на 100 классов, и тестировать каждый класс. Это существенно облегчило написание программного продукта.

Структурное программирование предполагает точно обозначенные управляющие структуры, программные блоки, отсутствие инструкций безусловного перехода (`GOTO`), автономные подпрограммы, поддержка рекурсии и локальных переменных.

Суть такого подхода заключается в возможности разбиения программы на составляющие элементы.

Также создавались *функциональные* (аппликативные) языки (Пример: Lisp — англ. *LISt Processing*, 1958) и*логические*языки (пример: Prolog — англ. *PROgramming in LOGic*, 1972).

Хотя структурное программирование, при его использовании, дало выдающиеся результаты, даже оно оказывалось несостоятельным тогда, когда программа достигала

определенной длины. Для того чтобы написать более сложную (и длинную) программу, нужен был новый подход к программированию.

ООП

В итоге в конце 1970-х и начале 1980-х были разработаны принципы объектно-ориентированного программирования. ООП сочетает лучшие принципы структурного программирования с новыми мощными концепциями, базовые из которых называются инкапсуляцией, полиморфизмом и наследованием.

Примерами объектно-ориентированных языков являются ObjectPascal, C++, Java, C# и др.

ООП позволяет оптимально организовывать программы, разбивая проблему на составные части, и работая с каждой по отдельности. Программа на объектно-ориентированном языке, решая некоторую задачу, по сути, описывает часть мира, относящуюся к этой задаче.