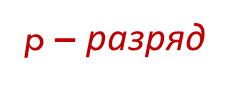
**Вопрос №1**

**Представление чисел в ЭВМ с фиксированной и плавающей точкой.**

1. ***Представление чисел с фиксированной точкой***

*Целые числа* хранятся в формате с фиксированной точкой (*точка фиксируется аппаратно после младшего разряда*).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **зн.** | **1p** | **2p** | **3p** | **4p** | **…** | **(n-1)p** | **np** | **“.”** |



***Этот маленький кусок пишу от себя***, так как сначала не понял, почему там под точку выделен бит и почему в некоторых случаях представления нужен знаковый бит, а иногда он отсутствует. *В общем мы может представлять целые числа с фиксированной точкой в беззнаковой или же в знаковой форме.*

Данная форма представления (которая выше в схеме) описывает целое знаковое число, а **точка фиксируется самой машиной**. То есть, когда мы представляем целое число *в беззнаковой форме* (это когда бит под знак отсутствует, *и числа в таком виде априори не могут быть отрицательными*), к примеру 255 из привычной нам десятичной системы, то получаем: 25510 =111111112. Здесь нет бита под знак, и как вы можете видеть, бита под точку тоже нет. Бит под точку не выделяется, машина сама фиксирует точку при представлении целого числа. И под неё БИТ НЕ ВЫДЕЛЯЕТСЯ. Не знаю зачем он запихнул эту точку в схему (которая выше), если под неё нет разряда. И на практике при переводе чисел как в знаковой, так и в беззнаковой форме *нигде нет этой точки в последнем разряде.* **Кароче, небольшой вывод**:

*Эту точку он просто показал, чтобы было видно, что машина её фиксирует. Но на практике, при записи чисел, «с фиксированной точкой» (хотя по сути дела это просто целые числа), точку в бит пихать не надо. Просто знайте, что она есть и всё.*

**Это информация просто для большего понимания темы, на экзамене так дословно писать не нужно. Просто сразу решил отсечь возникшие у вас вопросы.**

*Диапазон представление целых чисел в формате с фиксированной точкой*

* **зн 1 1 1 1 . . . 1 1 “.” Amax= 2n – 1**
* **зн 0 0 0 0 . . . 0 1 “.” Amin= 1**
* Всего можно представить чисел 2n
* Усредненная абсолютная погрешность представления числа есть среднее между минимально представимым числом и его минимальной потерей



* Относительная погрешность **δ = Δ / *А*min = 0,5**
* Погрешность достигает 50%

**Количество байт отводимых для хранения целых чисел** в формате с фиксированной точкой (на примере С++ )



* Целые числа могут быть представлены со знаком и без знака (***представляются в дополнительном коде***)



***Проблемы фиксированной точки***:

* Большая погрешность из – за ограниченного количества разрядов;
* Большие числа не помещаются в разрядную сетку – переполнение
* Малые числа воспринимаются как 0

Также следует помнить, что «1» и «-1» - это не одно и тоже. И здесь фишка не в том, чтобы просто поменять знаковый бит у положительного «1» и получить отрицательное. НЕТ !!! Так делать ЗАПРЕЩЕНО.

* **Знак +** кодируется как «**0»**, **Знак -** кодируется «**1»**
* Ноль кодируется как +0(0000), а **-0(1000) - этот код закрепляется за числом -8**
* Для четырех разрядов диапазон значений от -8 до +7.

**Достоинства формы представления с фиксированной точкой**

* Простота выполнения арифметических операций

**Недостатки**

* Ограничение длины разрядной сетки – *ограничение диапазона чисел   
  (и потеря точности)*

**Представление вещественных чисел**

***Вещественные числа могут быть записаны в виде:***

***Х = (+-)q(****+-****)Px ⋅ mx*** ,

*где* ***q*** *– основание системы счисления.*

***Р*** *– порядок, показывает на сколько*

*разрядов и в какую сторону сдвинута*

*запятая(«запятая плавает»)*

***m*** – мантисса , запись значения числа

***Например:***

***А10 = 239,745 = 0,239745 \* 103 = 2,39745 \* 102***

***= 239745 \* 10 -3***

***Одно и тоже число представлено по разному = неоднозначность представления***

Нормализованная форма числа

Для однозначности представления используют:

Нормальное число :

***q-1 ≤ |mx| < 1***

***0,239745 \* 103***

Нормализованное число :

***1 ≤ |mx| < 10 (для десятичной)***

***2,39745 \* 102***

\*В нормализованной мантиссе слева после запятой один знак, для двоичной системы это всегда 1.

***Машинное представление вещественных чисел с плавающей точкой***

Возможный вариант представления вещественных чисел в формате с плавающей точкой:

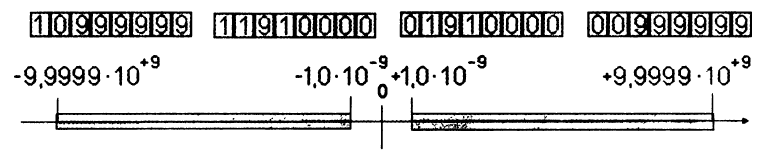
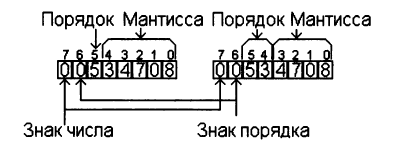
|  |  |  |  |
| --- | --- | --- | --- |
| +  **‾** | ***мантисса, mx*** | +  **‾** | ***порядок, px*** |

**Недостатки**:

– под знак порядка необходимо иметь знаковый разряд

– более сложные операции с порядками

**Диапазон представления чисел. Проблема «0»**



* Числа в диапазоне  не могут быть помещены в такую разрядную сетку.
* Ноль можно представить как: 00000000, 10000000, 01000000, 11000000

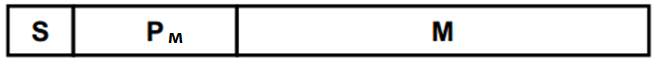
**Машинное представление вещественных чисел с плавающей точкой**

* Для исключения знака порядка он смещается на величину *Ксм* и получается машинный порядок Рм:

**Pм=Pн + Ксм,**

где Pн – порядок числа

* Ксм – смещение
* ***239745 \* 10-3 =2,39745 \* 102 (***Pм=2, Pн=-3*, Ксм=5)*



S-знак Pм- смещенный порядок, М-мантисса

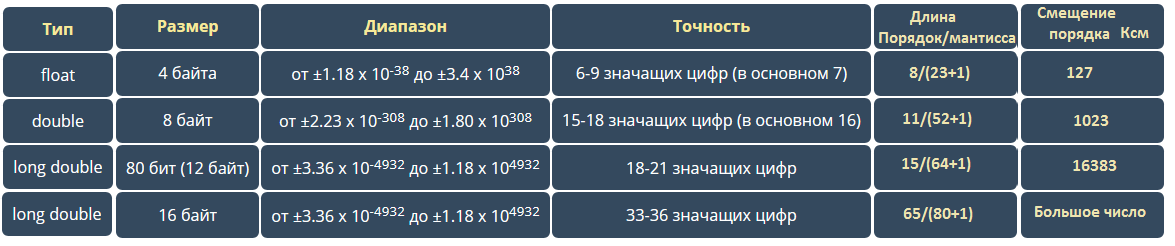
*Для двоичной системы смещение равно:* ***Ксм = 2N-1-1****,*

*где N – количество двоичных разрядов отведенных для*

*хранения порядка: при* ***N=8******Ксм =127, N=11******Ксм = 1023, N=15******Ксм =16383***

*Для однообразия способов представления чисел в форме с плавающей запятой Организацией IEEE (Institute of Electrical and Electronics Engineers)был разработан стандарт представления чисел* ***IEEE 754. (под порядок отводится 8, 11, 15 бит)***

*Для двоичной системы* целая часть мантиссы всегда единица, поэтому её можно не указывать, а соответствующий разряд всегда равный единице можно опустить.



***Пример размещения числа***

* Дано число -15,375(10)= -1111,011(2)
* Нормализуем: -1111,011(2)= -1,111011(2)\*23 (10) =

= -1,111011\*10011(2)

Смещение для порядка 8 бит:

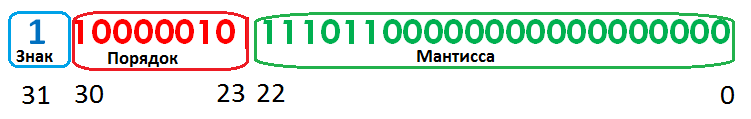
Ксм=2N-1-1=28-1-1=127(10)=0111 1111(2)

Машинный порядок Рм= Рн+ Ксм=127 +3=

=0111 1111(2) + 11(2) = 1000 0010(2)

Мантисса с отброшенной целой частью: 111011(2)

***Всё вместе***



***Преимущества представления чисел с плавающей точкой:***

* Относительная ошибка при представлении чисел в форме с плавающей точкой существенно меньше, чем в случае с фиксированной точкой.
* Больший диапазон изменения представляемых чисел.

*В языках высокого уровня используется такое представление:*

***(знак)(мантисса)Е(знак)(порядок)***

*НАПРИМЕР:* ***-5.35Е-2*** обозначает число ***-5.35\*10-2***

*Такое представление называется* ***научной нотацией.***

***Мои комментарии***: в моём ответе много схем, таблиц, так называемой «воды». Поэтому сразу говорю: учить это всё не надо. На экзамене достаточно просто расписать как представляются с целые числа и один два примера, и тоже самое с дробными. Ответ займёт у вас примерно 1-2 страницы. Данная «вода» нужна для большего понимания темы.

**Вопрос №2**

***Представление символов в ЭВМ – ASCII-соды, UNICODE.***

**ASCII:**

* 7 бит на символ.
* 32 (0-31) кода для команд + 95(33-27) для символов
* Этот стандарт был принят международной организацией по стандартизации *ISO (International Organization for Standardization)* как ISO 646:1991

Существует два вида кодировки символов **ASCII: 7 бит, 8 бит.**

К примеру, для ***ASCII 7 бит*** символу А соответствует код 41(16)= 65(10)

**ASCII- 8 бит:**

* Всего можно представить 256 (0-255)символов
* Из них:
  + первые 128 (0-127) кодов – стандартная 7-ми битная таблица ASCII
  + Вторые 128(128-255) кодов для кодирования **не латинских** символов
* Был принят международный стандарт ISO 8859 для второй половины кодов (для Unix-подобных ОС )

***Проблемы кодировок символов***

Операционные системы могут использовать различные кодировки символов, но есть проблемы:

* проблема ограниченности набора символов;
* проблема преобразования одной кодировки в другую;
* проблема дублирования шрифтов (символы на разных языках имели одинаковые коды).

***UNICODE***

* Стандарт состоит из двух основных частей: универсального набора символов ( *Universal character set, UCS*)

- семейства кодировок (*Unicode transformation format, UTF*).

**Универсальный набор символов**

* Универсальный набор символов **присваивает** каждому **символу номер** в виде неотрицательного целого числа, записываемого обычно в шестнадцатеричной форме с **префиксом U+.**
* Например, для кириллической буквы П код равен U+049F
* Область с номерами от **U+0000** до **U+007F** соответствует кодам латинских символов **ASCII**
* Под символы кириллицы выделены области с номерами U+0400 -- U+04FF, от U+2DE0 -- U+2DFF, от U+A640 -- U+A69F

**Семейство кодировок**

Семейство кодировок определяет как кодируются символы, представленные определенными номерами. Используются следующие кодировки: UTF 16, UTF8, UTF 32. То есть, **Буква «П» с номером U+041F кодируется в разных кодировках по-разному.**



***Связь UNICODE и таблицы ASCII***

* Для латинских символов соответствующих символам в таблице ASCII, номера и коды совпадают
* Например, латинская А имеет номер U+0041 и она кодируется как 0x0041

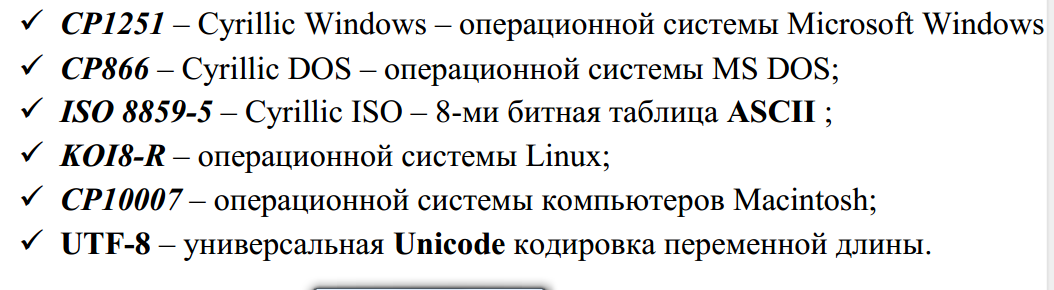
***Связь UTF 8 и UTF 16***

* Кодировки UTF-8 и UTF-16 обладают переменной длиной кода.
* В UTF-8 каждый юникод-символ может быть закодирован одним, двумя, тремя или четырьмя байтами. один байт
* В UTF-16 — двумя или четырьмя байтами.
* UTF-8 менее избыточная кодировка.
* Один и тот же файл:
  + в UTF-8 занимает 139Кб,
  + в UTF-16 уже 256Кб.

***Связь UTF-8 и UTF 32***

* ***UTF8 использует более гибкую схему***
* ***Для кодирования первой половины ASCII таблицы кодов используется 1 байт.***
* ***Для кодирования других символов используется переменное количество байт от одного до четырех.***
* ***UTF32 для кодирования использует 4 байта***

*Кодирование кириллицы:*

******

**Вопрос №3**

*Представление чисел в двоичном коде (прямой, обратный и дополнительный код). Сложение чисел с разными знаками в дополнительно коде.*

***Прямой двоичный код***

* Число записывается «как есть», дополняется нулями в старших разрядах до нужного размера.

00101101 45(10)

***Обратный код***

* Положительное число записывается «как есть», дополняется нулями в старших разрядах до нужного размера.
* Отрицательное число записывается **инвертированием** разрядов модуля числа.
* Старший бит определяет знак числа   
  (1 — отрицательное, 0 — положительное).

**Примеры** чисел в обратном коде:

01100111 103(10)

10011000 –103(10)

00000000 0(10)

11111111 –0(10)

***Дополнительный код десятичного числа***

* Дополнительный код положительного числа есть само число
* Дополнительный код отрицательного числа А есть число равное ***q N- A***, где q-основание системы счисления, N - количество разрядов в числе

***Дополнительный код двоичного числа***

* Положительное число записывается «как есть», дополняется нулями в старших разрядах до нужного размера.
* Отрицательное число записывается **инвертированием** разрядов модуля числа **и прибавлением 1**.
* Старший бит определяет знак числа   
  (1 — отрицательное, 0 — положительное).

**Примеры** чисел в дополнительном коде:

42 = *00101010*(2)

–42 =

00101010 (это просто 42, прямой код, число «-42» по модулю)

11010101 (это обратный код числа 42)

***11010110***(2) (это дополнительный код, полученный добавлением единицы к обратному).

***Сложение чисел с разными знаками в дополнительно коде.***

*Немного теории о сложении нулей и единиц в двоичной системе:*

1. *0 + 0 = 0*
2. *1 + 0 = 1 //* и наоборот
3. *1 + 1 = 10 // 102 = 210* (аналог в десятичной: 1 + 1 = 2)
4. *10 + 1 = 11 // 112 = 310* (аналог в десятичной: 2 + 1 = 3)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Десятичная форма | Двоичная форма | Прямой код | Обратный код | Дополнительный код |
| +12 | +1100 | 00001100 | 00001100 | 00001100 |
| -5 | -101 | 10000101 | 11111010 | 11111011 |

Выполним сложение в дополнительном коде:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | + | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
|  | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

Результат в дополнительном коде – 00000111. Поскольку знаковый разряд равен 0, результат положительный. Теперь можно восстановить алгебраическую запись результата. Он равен +111 (незначащие нули отброшены), или в десятичной форме +7. Проверка (+12-5=+7) показывает, что результат верный.

**Вопрос №4**

***Определение ОС. Основные функции ОС. Интерфейс пользователя и ОС (CLI, графический, программный API).***

*Определение ОС*

Операционная система — комплекс взаимосвязанных программ, предназначенных для

управления ресурсами компьютера и организации взаимодействия с пользователем.

Из чего состоит ОС:

* Ядро;
* Командный модуль;
* Комплекс драйверов;
* Сервисные утилиты;
* Оболочка.

Ядром операционной системы называют её основную часть, которая как раз и

обеспечивает запуск всех действий, организует работу других программ и работу

компонентов компьютерной системы.

Командный модуль — это программа, функцией которой является выполнение команд от

пользователя компьютеру.

Драйвер - это инструкция для управления операционной системой компьютерных

комплектующих.

Утилиты - дополнительные программные средства, которые взаимодействуют с ПО для

выполнения различных задач.

Оболочка операционной системы - это то, что пользователь видит на дисплее любого

компьютерного устройства.

*Основные функции*

В функции операционной системы входит:

— осуществление диалога с пользователем;

— ввод-вывод и управление данными;

— планирование и организация процесса обработки программ;

— распределение ресурсов (оперативной памяти и кэша, процессора, внешних

устройств);

— запуск программ на выполнение;

— всевозможные вспомогательные операции обслуживания;

— передача информации между различными внутренними устройствами;

— программная поддержка работы периферийных устройств (дисплея, клавиатуры,

дисковых накопителей, принтера и др.);

— организация среды взаимодействия и обмена информацией между работающими

программами.

***Интерфейс пользователя и ОС (CLI, графический, программный API).***

**Интерфе́йс по́льзователя**, он же по́льзовательский **интерфейс** (UI — англ. **user interface**) — **интерфейс**, обеспечивающий передачу информации между пользователем-человеком и программно-аппаратными компонентами компьютерной системы (ISO/IEC/IEEE 24765-2010).

**Интерфейс командной строки** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Command line interface,* ***CLI***) — разновидность [текстового интерфейса](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2%D1%8B%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81) (CUI) между человеком и компьютером, в котором инструкции компьютеру даются в основном путём ввода с клавиатуры текстовых строк (*команд*), в UNIX-системах возможно применение мыши. Также известен под названием *консоль*.

graphical user **interface**, GUI) — разновидность пользовательского **интерфейса**, в котором элементы **интерфейса** (меню, кнопки, значки, списки и т. ...)

**Графический интерфейс** пользователя является частью пользовательского **интерфейса** и определяет взаимодействие с пользователем на уровне визуализированной информации.

***API*** (программный интерфейс приложения, интерфейс прикладного программирования) (англ. application programming interface) — описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой. Обычно входит в описание какого-либо интернет-протокола, программного каркаса (фреймворка) или стандарта вызовов функций операционной системы. Часто реализуется отдельной программной библиотекой или сервисом операционной системы. Используется программистами при написании всевозможных приложений.

**Вопрос №5**

**Классификация ОС. Поколения ОС.**

1. ***Классификация ОС***

Операционные системы для персональных компьютеров делятся на:

– *одно- и многозадачные* (в зависимости от числа параллельно выполняемых прикладных

процессов);

– *одно- и многопользовательские* (в зависимости от числа пользователей, одновременно

работающих с операционной системой);

– *непереносимые* и *переносимые* на другие типы компьютеров;

– *несетевые и сетевые*, обеспечивающие работу в локальной вычислительной сети ЭВМ.

– *8-разрядные, 16-разрядные, 32-разрядные, 64-разрядные* (по разрядности кода,

Разрядность показывает, какую разрядность внутренней шины данных центрального

процессора способна поддержать операционная система);

– *однопроцессорные*(одноядерные) и *многопроцессорные*(многоядерные) (по количеству

процессоров (ядер) в компьютерной системе);

–  операционные системы *пакетной обработки*, *разделения времени*, *реального* времени

*(по типу доступа пользователя к ЭВМ).*

1. ***Поколения ОС***
2. *Нулевое поколение* (45-55е гг. ХХ в). (электронные лампы)

ОС на первых вычислительных машинах не было. Пользователи имели доступ к

машинному языку и все программы писались на машинных кодах.

1. *Первое поколение* (55-60е гг. ХХ в). (транзисторы)

Начало систем пакетной обработки. Операционные системы пакетной обработки – из

программ, подлежащих выполнению, формируется пакет (набор) заданий, вводимых в

ЭВМ и выполняемых в порядке очередности с возможным учетом приоритетности и

максимальной загрузки оборудования компьютера;

Пользователь не мог вести диалог с программой.

1. *Второе поколение* (1960-1965): (интегральные схемы)

Появление *многозадачных* (мультипрограммных) и *мультипроцессорных* (многопроцессорных) ОС, а также *операционных систем с разделением времени* - каждому пользователю системы

разделения времени предоставляется выносной терминал, с которого он может вести

диалог со своей программой.

1. *Третье поколение* (середина 65-х – конец 75-х гг.).

Операционные системы реального времени – используются на компьютерах,

управляющих технологическими процессами.

ОС реального времени обеспечивают реакцию на предусмотренные события за время, не

превышающее некоторое допустимое для данного события. Например, управление

ядерным реактором.

1. *Четвертое поколение* (сверхбольшие интегральные схемы конец 75-х – начало 90-х гг.).

Этот этап имеет следующие особенности:

- появление персонального компьютера и ОС к ним;

- распространение вычислительных сетей, появление и развитие сетевых ОС;

- большое внимание стало уделяться созданию ОС, ориентированных на

неподготовленного пользователя.

1. *Пятое поколение* (начало 90-х гг.).

Отличительные особенности этого этапа:

1. Широкое распространение и развитие персонального компьютера.

2. Развитие локальных и глобальных сетей, распространение Internetа.

3. Развитие графических пользовательских интерфейсов.

4. Создание мобильных цифровых устройств и ОС для них.

5. Появление многоядерных процессоров и ОС к ним.

**Вопрос №6**

**Определение программы. Понятие трансляции, интерпретации, компоновки**

*Что такое программа?*

*Программа* – это последовательность команд (операторов) определенного языка

программирования составленная для выполнения алгоритма (решения задачи).

Последовательность решения задачи:

*Постановка задачи* — формулируется цель решения задачи; проводится анализ

условий, при которых решается поставленная задача.

*Формальное построение модели задачи* — предполагает построение модели с

характеристиками, адекватными оригиналу, на основе какого-либо его физического

или информационного принципа; анализируется характер и сущность величин,

используемых в задаче.

*Построение математической модели задачи* — характеризуется математической

формализацией задачи, при которой существующие взаимосвязи между

величинами выражаются с помощью математических соотношений.

*Выбор и обоснование метода решения* — модель решения задачи реализуется на

основе конкретных приемов и методов решения. В большинстве случаев

математическое описание задачи трудно перевести на машинный язык. Выбор и

использование метода решения позволяет свести решение задачи к конкретному

набору машинных команд.

*Построение алгоритма* — на данном этапе составляется алгоритм решения задачи, в

соответствии с выбранным методом решения. Процесс обработки данных

разбивается на отдельные относительно самостоятельные блоки, определяется

последовательность выполнения этих блоков.

*Составление программы* — алгоритм решения переводится на конкретный язык

программирования.

*Отладка программы* — это специальный этап в разработке программы, состоящий в

выявлении и устранении программных ошибок, факт существования которых уже

установлен. Программные ошибки, как правило, делятся на три вида:

1. *Синтаксическая ошибка.* Неправильное употребление синтаксических конструкций.

В качестве примеров синтаксических ошибок в одном операторе можно назвать:

- пропуск необходимого знака пунктуации;

- несогласованность скобок;

- пропуск нужных скобок;

- неправильное формирование оператора;

- неверное образование имен переменных.

1. *Семантическая ошибка*. Нарушение семантики той или иной конструкции, например,

передача функции параметров, не соответствующих ее аргументам.

1. *Логическая ошибка*. Нарушение логики программы, приводящее к неверному

результату. Это наиболее трудный для &quot;отлова&quot; тип ошибки, ибо подобного рода

ошибки, как правило, кроются в алгоритмах и требуют тщательного анализа и

всестороннего тестирования.

*Решение задачи на компьютере и анализ результатов*. Теперь программу можно

использовать для решения поставленной задачи. Первоначально выполняется

многократное решение задачи на компьютере для различных наборов исходных

данных.

**Трансляция, интерпретация, компоновка**

* Машинные команды образуют сложный низкоуровневый машинный язык.
* Были разработаны (и разрабатываются) промежуточные высокоуровневые языки программирования, более удобные для человека.

Процесс перевода команд из языка высокого уровня в команды машинного языка называется трансляцией.

*Трансляция делится на:*

1. *Компиляцию*

* *Компиляция программы* - замена каждой команды программы на языке высокого уровня, эквивалентным набором команд на машинном языке и сохранение результата компиляции в отдельном файле
* После компиляции всей программы она запускается на выполнение.
* Компилятор - программа, которая осуществляет компиляцию.

1. Интерпретацию

* *При интерпретации* каждая команда языка высокого уровня переводится в эквивалентный набор команд машинного языка, после чего сразу выполняется (по -командно).
* Не требует составления всей программы на машинном языке.
* Программа, которая осуществляет интерпретацию, называется интерпретатором.
* Для повторного исполнения необходимо каждый раз

проводить интерпретацию программы

**Интерпретация**

1. ***Простой интерпретатор:***
   * читать код команды языка высокого уровня,
   * переводить в машинный язык
   * исполнять его сразу (Бэйсик);
2. ***Интерпретатор компилирующего типа:***
   * читать код всей программы
   * компилировать его полностью в промежуточный код (байт-код (Java) илиCIL (C#)),
   * последовательно переводить каждую команду промежуточного кода на машинный язык и сразу выполнять её*. (*С#, *Java,PHP, Pyton)*

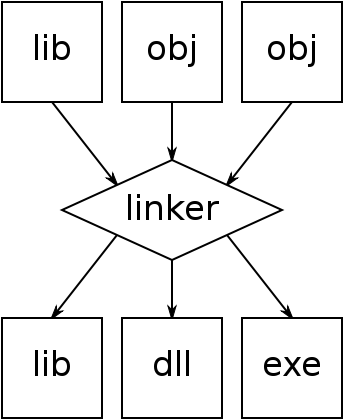
**Схема осуществления интерпретации**



***Компоновка (линовка)***

*Компиляция* относится к обработке файлов *исходного кода* (.c, .cc, или .cpp) и создании *объектных файлов* проекта. *На этом этапе не создается исполняемый файл*. Вместо этого компилятор просто транслирует высокоуровневый код в машинный язык. Например, если вы создали (но не скомпоновали) три отдельных файла, у вас будет три объектных файла, созданные в качестве выходных данных на этапе компиляции. Расширение таких файлов будет зависеть от вашего компилятора, например \*.obj или \*.o. Каждый из этих файлов содержит машинные инструкции, которые эквивалентны исходному коду*. Но вы не можете запустить эти файлы!* Вот тут за дело берётся компоновщик. *Используя полученные объектные файлы, он создаёт единый исполняемый / загрузочный модуль.* ***Также следует помнить, что:***

* Компоновщик, объединяя модули, пересчитывает относительные адреса относительно начала единого модуля.
* Компоновщик, который создает перемещаемый загрузочный модуль, называется редактором связей.
* Компоновщик обычно входит в состав интегрированной среды разработки

****

***Компоновка:***

* Компоновка зависит от типа создаваемого загрузочного модуля.
* Компоновка бывает статическая и динамическая.
* Статическая компоновка - модуль подключается к программе на этапе загрузки процесса в память, помещается в адресное пространство процесса и находится в памяти на время выполнения процесса.
* При статической компоновке (подключении) всех модулей (библиотек) к каждой исполняемой программе значительно возрастает объём её загружаемого модуля. При этом разные программы могут содержать одинаковые модули, которые дублируются в памяти