**Цель программной инженерии** – сокращение стоимости программ.

**Программа** – это объект разработки, который не является осязаемым (нельзя пощупать, взвесить и т. п.), доступен пониманию ЭВМ, для которой написан. логически упорядоченная последовательность команд, необходимых для решения определенной задачи. алгоритм, записанный на языке программирования.

**Текст программы** – полное законченное и детальное описание алгоритма на языке программирования.

**Программный продукт (ПП):** программа, работающая без авторского присутствия. Программный продукт исполняется, тестируется, конфигурируется без присутствия автора и сопровождается документацией.

**Программное обеспечение (ПО)** – совокупность программ системы обработки информации и программных документов, необходимых для эксплуатации этих программ (ГОСТ 19781-90)

**Программная инженерия** – это применение систематического, дисциплинированного и измеряемого подхода к разработке, эксплуатации и сопровождению программного обеспечения (ПО) с применением инженерных методов к разработке ПО.

**Жизненный цикл ПО** – непрерывный процесс с момента принятия решения о создании ПО до снятия его с эксплуатации.

**Основные типы стандартов**

**Корпоративные стандарты** разрабатываются крупными фирмами с целью повышения качества своей продукции. Создаются на основе собственного опыта компании, но с учетом требований мировых стандартов. Не сертифицируются, но являются обязательными для применения внутри корпорации.

**Отраслевые стандарты** действуют в пределах организаций некоторой отрасли (министерства). Разрабатываются с учетом требований мирового опыта и специфики отрасли. Являютсяобязательными для отрасли. Подлежат сертификации.

**Государственные стандарты (ГОСТы)** принимаются государственными органами и имеют силу закона. Разрабатываются с учетом мирового опыта или на основе отраслевых стандартов. Могут иметь как рекомендательный, так и обязательный характер. Для сертификации создаются государственные или лицензированные органы сертификации.

**Международные стандарты** разрабатываются специальными международными организациями на основе мирового опыта и лучших корпоративных стандартов. Имеют сугубо рекомендательный характер.

**Разработчики стандартов в области программной инженерии**

**ISO** – The International Standards Organization международная организация по стандартизации, работающая в сотрудничестве с **IEC – The International Electrotechnical Commission** – международной электротехнической комиссией

**IEEE Computer Society** – профессиональное объединение специалистов в области программной инженерии

**ACM** – Association for Computing Machinery – Ассоциация по вычислительной технике

**SEI** – Software Engineering Institute – Институт Программной Инженерии при университете КарнегиМелон

**PMI** – Project Management Institute – Международный Институт Проектного Менеджмента

**Классификация программного обеспечения:**

**Системное ПО** – комплекс программ, которые обеспечивают управление компонентами компьютерной системы: − управление ресурсами компьютера; − создание копий используемой информации; − проверка работоспособности устройств компьютера; − и др.

**Прикладные ПО** – предназначено для выполнения определённых пользовательских задач и рассчитанная на непосредственное взаимодействие с пользователем.

**Инструментальное ПО** – для автоматизации процесса разработки.

**Операционная система** — комплекс системных программ, расширяющий возможности вычислительной системы, обеспечивающий управление её ресурсами, загрузку и выполнение прикладных программ, взаимодействие с пользователями.

**Системы программирования** – системные программы, предназначенные для разработки программного обеспечения.

**Система программирования**: комплекс программных средств, предназначенных для автоматизации процесса разработки, отладки программного обеспечения и подготовки программного кода к выполнению

**Состав системы программирования** трансляторы компоновщики отладчики профилировщики программные библиотеки редакторы кода системы поддержки версий и пр.

**Язык программирования** формальная знаковая система, предназначенная для записи компьютерных программ. Знаковая система определяет набор лексических, синтаксических и семантических правил написания программы (программного кода). Язык программирования представляется в виде набора спецификаций, определяющих его синтаксис и семантику.

**ASCII (American Standard Code for Information Interchange)** — американский стандартный код для обмена информацией. ASCII — 8-битная кодировка для представления десятичных цифр, латинского и национального алфавитов, знаков препинания и управляющих символов. Таблица кодов ASCII делится на две части: Международным стандартом является первая половина таблицы, т.е. символы с номерами от 0 (00000000), до 127 (01111111). К концу 1980-х годов стандартом стали 8-битные кодировки.

**Переносимый набор символов (portable character set)** – набор из 103 символов, которые (стандарт POSIX) должны присутствовать в любой используемой кодировке.

**POSIX (англ. Portable Operating System Interface — переносимый интерфейс операционных систем)** — набор стандартов, описывающих интерфейсы между операционной системой (ОС) и прикладной программой (системный API), библиотеку языка C и набор приложений и их интерфейсов.

**Windows-1251** — набор символов и кодировка, являющаяся стандартной 8-битной кодировкой для русских версий Microsoft Windows до 10-й версии

**Юникод** – стандарт кодирования символов, позволяющий представить знаки почти всех письменных языков, состоит из 2х разделов: − UCS – universal character set (универсальный набор символов); − UTF – Unicode transformation format (семейство кодировок).

**UTF-8** — представление Юникода, обеспечивающее совместимость со старыми системами, использовавшими 8-битные символы.

**Алгоритм кодирования в UTF-8:** 1) определить количество октетов (октет: 8 битов или 1 байт) – т.е. в какой диапазон значений попадает количество значащих символов (7, 11, 16, 21, 26, 31); 2) подготовить старшие биты первого октета: a. 0xxxxxxx для одного октета; b. 110xxxxx – для двух; c. 1110xxxx – для трех и т.д.. d. 10xxxxxx – для остальных октетов; 3) заполнить оставшиеся биты (выше обозначены как x) в октетах кодом символа Юникода в двоичном виде. Начать с младших битов, поставив их в младшие биты последнего октета кода. И так далее, пока все биты кода символа не будут перенесены в свободные биты октетов.

В **UTF-16** символы кодируются двухбайтовыми словами (16 битов) с использованием всех возможных диапазонов значений (от 0 до FFFF16).

**Стандарт языка** **набор спецификаций,** определяющих его синтаксис, семантику, может исторически развиваться.

**Реализация языка** **программные средства**, которые обеспечивают определенный вариант стандарта языка (производитель, марка, версия; могут иметь ошибки)

**Способы реализации языков** компилируемые интерпретируемые

**Алфавит языка программирования** – набор символов, разрешенных к использованию языком программирования. Основывается на одной из кодировок.

**Базовый набор символов исходного кода**: 1) строчные и прописные буквы латинского и национального алфавитов 2) цифры 3) знаки операций 4) символы подчеркивания \_ и пробельные символы 5) ограничители и разделители 6) специальные символы

**Компилятор** – программа, преобразующая исходный код на одном языке программирования в исходный код на другом языке. Результат – объектный модуль.

**Набор символов времени трансляции:** текст программы на языке программирования хранится в исходных файлах и основан на определенной кодировке символов

**Набор символов времени выполнения:** символы, отображаемыми в среде выполнения. Любые дополнительные символы зависят от локализации

**Лексемы** идентификаторы; ключевые (зарезервированные) слова; знаки операций; константы; разделители (скобки, знаки операций, точка, запятая, пробельные символы и т.д.).

**Идентификатор** – имя компонента программы (переменной, функции, метки, типа и пр.), составленное программистом по определенным правилам.

**Этапы и цели разработки программы:**

**1. Постановка задачи.** • определение функциональных возможностей программы; • подготовка технического задания

**2. Выбор метода решения.** • определение исходных и выходных данных, ограничений на них; • выполнение формализованного описания задачи; • построение математической модели, для решения на компьютере.

**3. Разработка алгоритма решения задачи.** • выполняется на основе ее математического описания; • полное и точное описание, определяющее вычислительный процесс, ведущий от начальных данных к искомому результату.

**4. Написание программы на языке программирования (кодирование)** • запись алгоритма на языке программирования.

**5. Ввод программы в компьютер** • подготовка исходного кода программы в виде текстового, который поступает на вход транслятора.

**6. Трансляция** • преобразование исходного кода с одного языка программирования в семантически эквивалентный код на другом языке; • получение объектного модуля.

**7. Компоновка** • объединение одного или нескольких объектных модулей программы и объектных модулей статических библиотек в исполняемую программу; • связывание вызовов функций и их внутреннего представления (кодов), расположенных в различных модулях; • получение исполняемого (загрузочного) файла.

**8. Выполнение** • выполнение исполняемого файла программы на целевой машине.

**9. Тестирование** • устранение ошибок в программе.

**10. Отладка** • обнаружение, локализация и устранение ошибок.

**11. Документирование** • создание пользовательской документации.

**12.** **Эксплуатация** • выполнение в предназначенной для этого среде в соответствии с пользовательской документацией

**13.** **Модификация (Реинжиниринг)** • внесение изменений в ПО в целях исправления ошибок, повышения производительности или адаптации к изменившимся условиям работы или требованиям.

**14.** **Снятие с эксплуатации** • завершение жизненного цикла ПП и изъятие его из эксплуатации.

**Алгоритм** точное предписание, определяющее вычислительный процесс, ведущий от начальных данных к искомому результату. конечный набор правил, который определяет последовательность операций для решения конкретного множества задач и обладает пятью важными чертами: конечность, определённость, ввод, вывод, эффективность.

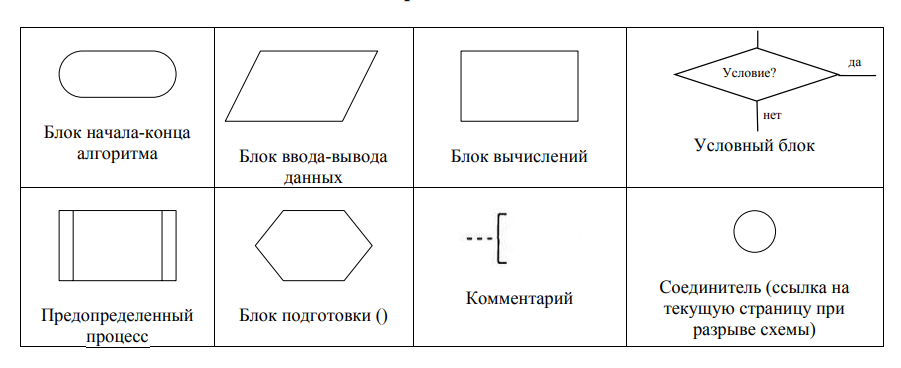
**Программирование** — это также раздел прикладной математики, разрабатывающий методы использования вычислительных машин для реализации алгоритмов

**Свойства алгоритмов** дискретность (возможность разбиения на шаги); понятность (ориентирован на исполнителя); определенность (однозначность толкования инструкций); конечность (возможность получения результата за конечное число шагов); массовость (применимость к некоторому классу объектов); эффективность (оптимальность времени и ресурсов, необходимых для реализации алгоритма).

**Процесс алгоритмизации** разложение всего вычислительного процесса на отдельные шаги, установление взаимосвязей между отдельными шагами алгоритма и порядка их следования, полное и точное описание содержания каждого шага, проверка правильности составленного алгоритма

**Способы описания алгоритмов** словесно-формульный (на естественном языке); графический (структурный или блок-схемой); использование псевдокода (специальных алгоритмических языков); с помощью сетей Петри; программный.

**Основные элементы схем алгоритма**



**Типы процессов**

**Линейные процессы**. направление вычислений не зависит от значения исходных данных и получаемых в результате решения задачи промежуточных результатов.

**Разветвляющиеся процессы**. вычислительные процессы, в которых в зависимости от значения некоторого признака проводятся вычисления по одному из нескольких возможных направлений, называются ветвящимися (разветвляющимися).

**Кодирование**: запись разработанного алгоритма в виде программы на выбранном языке программирования. Результатом этапа является исходный код программы на ЯП.

**Текстовый редактор компонента системы программирования (или IDE)** – программа, позволяющая подготовить исходный код программы

**Интегрированная среда разработки (integrated development environment - IDE)** набор инструментов для разработки и отладки программ, имеющий общую интерактивную графическую оболочку, поддерживающую выполнение всех основных функций жизненного цикла разработки программы

**Компилятор (транслятор)** – программа, преобразующая исходный код на одном языке программирования в исходный код на другом языке; результат – объектный модуль.

**Интерпретатор** – разновидность транслятора. Переводит и выполняет программу с языка высокого уровня в машинный код строка за строкой.

**Компоновщик (linker, редактор связей)** – программа, принимающая один или несколько объектных модулей и формирующая на их основе загрузочный модуль.

**Загрузочный код** – результат работы компоновщика. Один файл загрузочного кода – загрузочный модуль.

**Загрузчик (loader)** – программа, обычно входящая в состав операционной системы, предназначенная для запуска процесса операционной системы на основе загрузочного модуля.

**Отладка программы** – процесс поиска, локализации и устранения ошибок в программе.

**Отладчик (debugger)** – компонента системы программирования (или IDE) – программа, позволяющая контролировать ход выполнения программы (приостанавливать, выполнять пошагово), просматривать и изменять области памяти и.т.п.

**Тест** – это набор конкретных значений исходных данных, при которых известен ожидаемый результат работы программы.

**Документирование** – создание текстовых и графических материалов по использованию программы в помощь пользователям и разработчикам (общее описание возможностей программы, техники использования, типовые примеры и т.д.).

**Реинжиниринг** – это модификация программного продукта при необходимости исправить ошибки, выявленные в процессе эксплуатации, модернизировать или адаптировать программу к изменившимся требованиям.

**Примеры IDE:** Visual Studio, NetBeans, Eclipse, Embarcadero Delphi и пр.

**JDeveloper** — бесплатная интегрированная среда разработки ПО. Для разработки на языках программирования Java, JavaScript, BPEL, PHP, SQL, PL/SQL; на языках разметки HTML, XML.

**NetBeans** — свободная интегрированная среда разработки ПО на языках программирования Java, Python, PHP, JavaScript, C, C++, Ада и др.

**Eclipse** — свободная интегрированная среда разработки модульных кроссплатформенных приложений.

**Oracle Sun Studio** — интегрированная среда разработки для языков программирования Си, C++ и Фортран

**Embarcadero Delphi** — интегрированная среда разработки ПО для Microsoft Windows, Mac OS, iOS и Android на языке Delphi (раньше Object Pascal). Embarcadero Delphi является частью пакета Embarcadero RAD Studio

**Microsoft Visual Studio** — линейка продуктов компании Microsoft, включающих интегрированную среду разработки ПО и другие инструменты. Для разработки консольных приложений, игр, приложений с графическим интерфейсом, веб-сайтов, веб-приложений, веб-служб как в нативном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight.

**ECMAScript** — это встраиваемый расширяемый язык программирования, основа для построения других скриптовых языков. Стандартизирован международной организацией Ecma в спецификации ECMA-262. Расширениями языка являются JavaScript (Netscape), JScript (Microsoft) и ActionScript.

**Unix («UNIX» – зарегистрированная торговая марка The Open Group)** – семейство переносимых, многозадачных и многопользовательских операционных систем, которые основаны на идеях оригинального проекта AT&T Unix, разработанного в 1970-х годах в исследовательском центре Bell Labs Кеном Томпсоном, Деннисом Ритчи и другими.

**Linux (GNU/Linux)** – семейство Unix-подобных операционных систем на базе ядра Linux, включающих тот или иной набор утилит и программ проекта GNU (проект по разработке свободного программного обеспечения, запущенный известным программистом и сторонником СПО Ричардом Столлманом 27 сентября 1983 года в Массачусетском технологическом институте.

**Интерфейс вызовов функций в Windows** – доступ к системным ресурсам осуществляется через целый рад системных функций. Совокупность таких функций называется прикладным программным интерфейсом или API (Application Programming Interfase). Для взаимодействия с Windows приложение запрашивает функции API, с помощью которых реализуются все необходимые системные действия, такие как выделение памяти, вывод на экран, создание окон и т.п.

В Windows два типа многозадачности: основанный на процессах и основанный на потоках. **Процесс** – это программа, которая выполняется. При многозадачности такого типа две или более программы могут выполняться параллельно. **Поток** – это отдельная часть исполняемого кода. В многозадачности данного типа отдельные потоки внутри одного процесса также могут выполняться одновременно.

**Стандарт POSIX** – переносимый интерфейс операционных систем) — набор стандартов, описывающих интерфейсы между операционной системой и прикладной программой (системный API), библиотеку языка C и набор приложений и их интерфейсов.

**Парадигма программирования** – это совокупность идей и понятий, определяющих стиль написания компьютерных программ (подход к программированию).

**Декларативное программирование**: Программа = описание действий, которые необходимо выполнить компилятору для получения результата. Отвечает на вопрос что надо выполнить.

**Функциональное программирование**: Программа = система определений и функций, описывающих что нужно вычислить, а как это сделать – решает транслятор; последовательность действий не прослеживается. Раздел дискретной математики. Основой функционального программирования является лямбда-исчисление

**Объектно-ориентированное программирование**: Программа = несколько взаимодействующих объектов + функциональность (действия и данные распределяются между этими объектами).

**Распределённое (параллельное) программирование**: Программа = совокупность описаний процессов, которые могут выполняться как параллельно (при наличии нескольких процессоров), так и в псевдопараллельном режиме (при наличии одного процессора).

**Логическое программирование**: Программа = система определений вида «условие => новый факт». Программа представляет собой описание фактов и правил вывода в некотором логическом исчислении. Результат, (который часто записывается как вопрос), получается системой путем логического вывода. Раздел математической логики.

**Распределённое (параллельное) программирование**: Программа = совокупность описаний процессов, которые могут выполняться как параллельно (при наличии нескольких процессоров), так и в псевдопараллельном режиме (при наличии одного процессора).

**Визуальное программирование**: Программа = способ создания программы для ЭВМ путём манипулирования графическими объектами вместо написания её текста. Визуальное программирование позволяет программировать на уровне алгоритмов, а не программного кода. Пакет визуального программирования генерирует, написанный на языках программирования (1GL, 2Gl, 3GL), на основании составленной программистом «блок-схемы» в автоматическом режиме.

**Аспектно-ориентированное программирование**: Программа = к уже существующему коду добавляется дополнительного поведение, так называемой сквозной функциональности

**Индекс TIOBE** – индекс, который оценивает популярность языков программирования, основываясь на количестве поисковых запросов, содержащих название языка. Расчет индекса происходит ежемесячно.

**Языки классифицируют по важнейшим признакам:** − эволюционным – поколения языков (1GL, 2GL, 3GL, 4GL, 5GL, ...); − функциональным – по назначению, исполняемым функциям (описательные, логические, математические); − уровню языка – то есть уровню обобщения в словах-операторах языка (низкого, среднего, высокого, ...); − области применения –системные, сетевые, встроенные и пр.

**Поколения языков (Generations of Languages)**

Каждое из последующих поколений по своей функциональной мощности качественно отличается от предыдущего.

• 1GL - первое поколение: Машинные языки. Появились в середине 40-х годов XX века.

• 2GL - второе поколение: Ассемблеры. Фактически это те же машинные языки, но более красиво «обернутые». Появились в конце 50-х годов XX века

• 3GL - третье поколение: Процедурные языки. Появились в начале 60-х годов XX века. К этому поколению относят универсальные языки высокого уровня, с помощью которых можно решать задачи из любых областей (например, Algol-60).

• 4GL - четвертое поколение: Языки поддержки сложных структур данных (например, SQL). Появились в конце 60-х годов XX века.

• 5GL - пятое поколение: Языки искусственного интеллекта (например, Prolog). Появились в начале 70-х годов XX века.

• 6GL - шестое поколение: Языки нейронных сетей (самообучающиеся языки). Исследовательские работы в этой области начались в середине 80-х годов XX века.

**Методология** включает в себя модель вычислений для данного стиля.

**Методология разработки программного обеспечения** – совокупность методов, применяемых на различных стадиях жизненного цикла программного обеспечения

**Неструктурное программирование** характерно для наиболее ранних языков программирования. Сложилось в середине 40-х с появлением первых языков программирования. Основные признаки: − строки как правило нумеруются; − из любого места программы возможен переход к любой строке;

В **сентенциальном стиле** каждый шаг программы проверяет весь объем текстовой информации на соответствие образцу, находит и, согласно найденному правилу преобразования, производит преобразование.

В **программировании от событий** условие состоит в том, что в системе произошло некоторое событие, которое влечет обработку данного действия обработчиком события.

**Поток входных данных** — это последовательность нажимаемых клавиш и принимаемые (в параллельном режиме) телефоном сигналы.

**Глобальными переменными (памятью)** являются все настройки, адресная книга, списки вызовов, SMS сообщения, флаги (информация) о пропущенных вызовах или полученных SMS сообщениях и др

**Императивное программирование** (от греч. imper — действие) предполагает, что программа явно описывает алгоритм решения конкретной задачи (действия исполнителя), т.е. описывает как решать поставленную задачу.

**Декларативное программирование** (лат. declaratio – объявление, подход возник в 60-х годах) – это предварительная реализация «решателя» для целого класса задач. Тогда для решения конкретной задачи этого класса достаточно декларировать в терминах данного языка только её условие: (исходные данные + необходимый вид результата) «Решатель» сам выполняет процесс получения результата, реализуя известный ему алгоритм решения.

**Структурное программирование** – методология и технология разработки программных средств, основанная на трёх базовых конструкциях: − следование; − ветвление; − цикл.

**Технология структурного программирования базируется на следующих методах**:

**− нисходящее проектирование;**

Нисходящее проектирование программы состоит в процессе формализации от самой внешней синтаксической конструкции алгоритма к самой внутренней; в движении от общей формулировки алгоритма к частной формулировке, составляющей его действия;

**− пошаговое проектирование;**

Пошаговое проектирование состоит в том, что на каждом этапе проектирования в текст программы вносится только одна конструкция языка.

**− структурное проектирование (программирование без goto);**

Структурное проектирование заключается в замене словесной формулировки алгоритма на одну из синтаксических конструкций – последовательность, условие или цикл. Использование оператора безусловного перехода goto запрещается из принципиальных соображений;

**− одновременное проектирование алгоритма и данных;**

Одновременное проектирование алгоритма и структур данных. При нисходящей пошаговой детализации программы необходимые для работы структуры данных и переменные появляются по мере перехода от неформальных определений к конструкциям языка, то есть процессы детализации алгоритма и данных идут параллельно.

**− модульное проектирование;**

**− модульное, нисходящее, пошаговое тестирование.**

Цель (результат) = действие + цель (результат) вложенной конструкции.

**Последовательность действий, связанных результатом**. − представить действие в виде последовательности шагов; − между различными шагами существуют связи через общие переменные, результат выполнения шага и последующие шаги используют этот результат. **Программирование без goto**. Операторы continue, break и return: служат для более «мягкого» нарушения структурированной логики выполнения программы: − continue – переход завершающую часть цикла; − break – выход из внутреннего цикла; − return – выход из текущего модуля (функции).

**Консольное приложение** использует окно консоли Windows для отображения выходных данных и приема данных, вводимых пользователем.

**Обозреватель решений**. В обозревателе решений отображаются графическое представление иерархии файлов и папок в проекте, оно находится справа.

**Файловую структуру решения;**

1) **Внешние зависимости** – содержит ссылки на все модули, которые использует программа.

2)**Файлы заголовков** – содержит файлы кода С++ с расширением h.

3)**Исходные файлы** – содержит файлы кода С++ с расширением срр.

4)**Файлы ресурсов** – содержит файлы, непосредственно не относящиеся к языку С++, но необходимые для работы приложения. Например, мультимедийные файлы.

В результате компиляции создается файл с расширением **obj** – объектный модуль программы. Для каждого файла .cpp будет по одному, или более "единиц компиляции". Они создаются на этапе компиляции при создании проекта.

**Exe** – исполняемый файл проекта. Исполняемый файл — набор инструкций, который заставляет [компьютер](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80) выполнить определённую задачу. В отличие от [текстового файла](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2%D1%8B%D0%B9_%D1%84%D0%B0%D0%B9%D0%BB), который рассчитан на чтение человеком, исполняемый файл рассчитан на чтение (и выполнение) [процессором](https://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BD%D1%82%D1%80%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80).

Расширение **LOG**, как подсказывает его название, связано с лог-файлами работающего компьютера. Как правило, они представляют собой текстовые файлы, которые содержат информацию о запущенных или используемых программ, а также временные метки.

Главное различие состоит в назначении: конфигурация **Debug** предназначена для компиляции на этапе разработки и отладки программы, а **Release** - для сборки программы и последующего её использования пользователями программы. В конфигурации Release удаляется отладочная информация из исполняемого файла. Это приводит к уменьшению размера исполняемого файла (обычно в несколько раз).

**Таблицы кодировки** - таблицы, где каждому символу, буквам, цифрам, а также специальным знакам присвоен уникальный номер - код символа.