## 1. Современные подходы к разработке программного обеспечения (ПО): понятие конструирования ПО, основные этапы конструирования ПО.

**Понятие конструирования программного обеспечения**



**Конструирование** – единственный процесс, который выполняется всегда – это процесс создания какого-нибудь объекта, может включать в себя некоторые аспекты планирования, проектирования и тестирования.

высокоуровневое проектирование;

* детальное проектирование;
* кодирование и отладка;
* блочное тестирование;
* интеграционное тестирование;
* интеграция;
* тестирование системы;
* корректирующее сопровождение.

или

ПО,

архитектуры

разработка

определение проблемы (анализ); выработка требований;

создание плана конструирования;









***Компоненты разработки ПО***:

## 2. Современные подходы к разработке программного обеспечения: понятие конструирования ПО; результат конструирования; решения, которые принимаются при конструировании.

**Конструирование** – часть процесса разработки ПО. В зависимости от размера проекта на конструирование обычно уходит 30–80% общего времени работы.

Конструирование занимает центральное место в процессе разработки ПО. Требования к приложению и его архитектура разрабатываются до этапа конструирования, что гарантирует его эффективность. Тестирование системы выполняется после конструирования и служит для проверки его правильности.

**Результат конструирования** – исходный код – часто является единственным верным описанием программы.

* **конструирование** – главный этап разработки ПО, без которого не обходится ни один проект;
* **основные этапы конструирования**: ***детальное проектирование, кодирование, отладка, интеграция и тестирование приложения разработчиками*** (блочное тестирование и интеграционное тестирование);
* **конструирование** часто называют «кодированием» и

«программированием»;

* от качества конструирования во многом зависит качество ПО;
* компетентность в конструировании ПО определяет то, насколько хорошим программистом вы являетесь.

***Основные решения, которые принимаются при конструировании:***

* выбор языка программирования;
* конвенции программирования;
* выбор технологий;
* выбор основных методик конструирования.

## 3. Система программирования: определение, состав, назначение. Классический жизненный цикл разработки ПО. Текстовый редактор. Интегрированная среда разработки. Программный продукт. Общая схема преобразования исходного кода в процесс операционной системы.

**Определение**: комплекс программных средств, предназначенный для реализации программного обеспечения (написания и отладки программного кода).

**Состав:**

* **трансляторы** - программа, преобразующая исходный код на одном языке программирования в исходный код на другом языке, **компоновщики** - программа, принимающая один или несколько объектных модулей и формирующая на их основе загрузочный модуль,
* **отладчики** - программа, позволяющая контролировать ход выполнения программы (приостанавливать, выполнять пошагово), просматривать и изменять области памяти,
* **профилировщики** - программа, позволяющая оптимизировать код программы (устранять утечки памяти, оптимизировать циклы, анализировать производительность),
* **программные библиотеки**,
* **редакторы кода**,
* **системы поддержки версий**.

**Назначение:** система программирования представляет собой совокупность средств разработки программ, обеспечивающих автоматизацию составления и отладки программ пользователя.

**Классический жизненный цикл разработки программного обеспечения:**

**Жизненный цикл программного обеспечения** (ПО) — период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент его полного изъятия из эксплуатации. Этот цикл — процесс построения и развития ПО.

****

**Текстовый редактор:** компонента системы программирования - программа, позволяющая подготовить исходный код программы.

**Интегрированная система разработки:** набор инструментов для разработки и отладки программ, имеющий общую интерактивную графическую оболочку, поддерживающую выполнение всех основных функций жизненного цикла разработки программы.

**Интегрированная среда разработки** (IDE, Integrated development environment): редакторы кода, транслятор, компоновщик, отладчик, система поддержки версий. Примеры: Visual Studio, NetBeans, Eclipse, Embarcadero Delphi и пр.

**Программный продукт:** программа, работающая без авторского присутствия; исполняется, тестируется, конфигурируется без присутствия автора; сопровождается документацией.

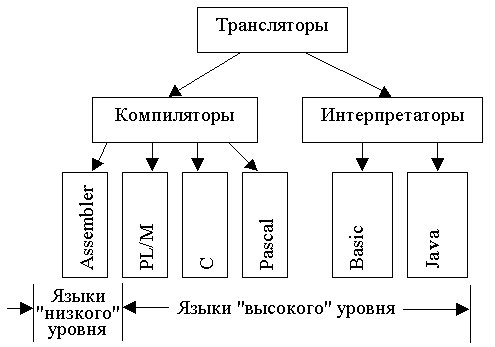
***Общая схема преобразования исходного кода в процесс операционной системы:***



**Транслятор**: программа, преобразующая исходный код на одном языке программирования в исходный код на другом языке.

Пример: Ассемблер: транслятор с исходного кода на языке ассемблера в программу на машинном языке (язык, который может интерпретироваться процессором).

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



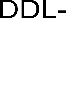
## 4. Интегрированная среда разработки: определение, состав, назначение. Транслятор: виды трансляторов. Транслятор: определение, назначение, примеры. Интерпретатор: определение, назначение, примеры.

**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.

***Примеры IDE*** (визуальные среды): Eclipse, Microsoft Visual Studio, NetBeans, Qt Creator,

**Интегрированная среда разработки**, используется программистами для разработки создания и тестирования программ или программного обеспечения.

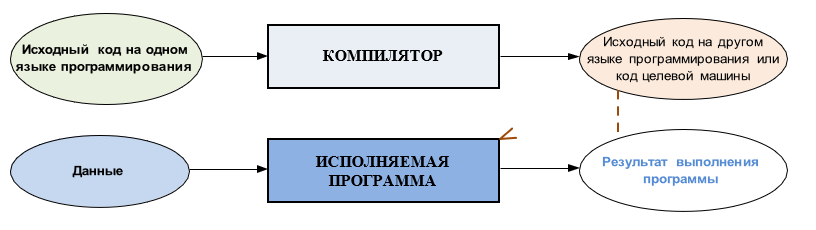
Под **отладкой** программы понимается процесс испытания работы программы и исправления обнаруженных при этом ошибок. Обнаружить ошибки, связанные с нарушением правил записи программы **на** **языке** **программирования** (синтаксические и **семантические ошибки), помогает используемая система программирования.**



**Трансля́тор** — программа или техническое средство, выполняющее трансляцию программы. Трансля́ция програ́ммы — преобразование программы, представленной на одном из языков программирования, в программу на другом языке.

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

Схема работы транслятора: Исходный код на одном языке программирования 🡪 Компилятор (транслятор) 🡪 Исходный код на другом языке программирования или код целевой машины 🡪 Результат выполнения программы 🡨 ИСПОЛНЯЕМАЯ ПРОГРАММА 🡨 Данные



## 5. Язык программирования: определение, назначение, примеры. Исходный код. Язык ассемблера. Транслятор: определение, назначение, примеры. Алфавит языка. Ассемблер. Объектный код, объектный модуль. Общая схема преобразования исходного кода в процесс операционной системы.

**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.

Примеры: процедурно-ориентированные, объектно-ориентированные, декларативные, операторные, функциональные, скриптовые, проблемно-ориентированные, машинно-зависимые (ассемблеры) и т.п.

**Исходный код (исходная программа)**: текст программы, написанный на языке программирования.

Программа на исходном языке (исходный код) готовится с помощью текстовых редакторов (программ, позволяющих подготовить исходный код программы) и в в

иде текстового файла или раздела библиотеки поступает на вход транслятора.

**Язык ассемблера:** машинно-ориентированный язык программирования (для конкретной архитектуры компьютера, команды которого соответствуют машинным командам)

**Транслятор:** программа, преобразующая исходный код на одном языке программирования в исходный код на другом языке.

**Трансляторы** (компиляторы) полностью обрабатывают весь исходный текст программы (исходный код). Они просматривают его в поисках синтаксических ошибок (иногда несколько раз), проверяют семантические правила языка.

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

**Примеры:** диалоговый, синтаксически-ориентированный, однопроходной, многопроходной, оптимизирующий, тестовый, обратный

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

**Ассемблер:** транслятор с исходного кода на языке ассемблера в программу на машинном языке (язык, который может интерпретироваться процессором).

**Объектный код:** результат работы транслятора. Один файл объектного кода – **объектный модуль**.

**Общая схема преобразования исходного кода в процесс операционной системы.**

## 6. Язык программирования: определение, назначение, примеры. Исходный код. Язык ассемблера. Транслятор: определение, назначение, примеры. Алфавит языка. Ассемблер. Объектный код, объектный модуль. Общая схема преобразования исходного кода в процесс операционной системы.

**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.

Примеры: процедурно-ориентированные, объектно-ориентированные, декларативные, операторные, функциональные, скриптовые, проблемно-ориентированные, машинно-зависимые (ассемблеры) и т.п.

**Исходный код (исходная программа)**: текст программы, написанный на языке программирования.

Программа на исходном языке (исходный код) готовится с помощью текстовых редакторов (программ, позволяющих подготовить исходный код программы) и в в

иде текстового файла или раздела библиотеки поступает на вход транслятора.

**Язык ассемблера:** машинно-ориентированный язык программирования (для конкретной архитектуры компьютера, команды которого соответствуют машинным командам)

**Транслятор:** программа, преобразующая исходный код на одном языке программирования в исходный код на другом языке.

**Трансляторы** (компиляторы) полностью обрабатывают весь исходный текст программы (исходный код). Они просматривают его в поисках синтаксических ошибок (иногда несколько раз), проверяют семантические правила языка.

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

**Примеры:** диалоговый, синтаксически-ориентированный, однопроходной, многопроходной, оптимизирующий, тестовый, обратный

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

**Ассемблер:** транслятор с исходного кода на языке ассемблера в программу на машинном языке (язык, который может интерпретироваться процессором).

**Объектный код:** результат работы транслятора. Один файл объектного кода – **объектный модуль**.

**Общая схема преобразования исходного кода в процесс операционной системы.**

## 7. Иерархическая структура компонентов в Visual C++: глобальный контейнер, компоненты, входящие в его состав. Структура проекта Visual C++.

***Иерархическая структура компонентов в Visual C++.***

Глобальный контейнером (компонент, включающий в себя другие компоненты) является ***Решение***. Решение может содержать один или несколько проектов.

Проекты являются независимыми компонентами. Они имеют собственную структуру, состоящую из четырех основных каталогов:

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

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

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

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

Код программного проекта может иметь сложную структуру и состоять из нескольких файлов исходного кода и конфигурационных файлов.

## 8. Иерархическая структура компонентов в Visual C++: многофайловый проект, предварительно откомпилированные заголовки: назначение, создание, применения. Пример (C++).

**Страницы свойств проекта. Разделы свойств. Параметры.**

**Общие** – ключ **Уровень предупреждений** позволяет отключить все предупреждения (/W0), либо ужесточить уровень проверок и считать все предупреждения ошибками (/Wall).

**Предварительно откомпилированные заголовки** позволяют их включить/отключить, определить ***имя*** создаваемого предварительно откомпилированного заголовочного файла и местоположение для полученного предварительно откомпилированные заголовки выходного файла (с расширением pch). Также можно настроить имена и папки, в которых будут размещаться различные **Выходные файлы**.

**Командная строка** компилятора С++ отображает, с какими параметрами (ключами) выполняется текущая компиляция.

Раздел **Компоновщик** отображает и позволяет изменить текущие настройки и ключи компоновки.

Указывает компилятору создать предварительно скомпилированный файл заголовка (PCH), представляющий состояние компиляции в определенной точке.

## Синтаксис

**/Yc**  
Имя файла**/Yc**

Предписывает компилятору использовать существующий файл предкомпилированного заголовка ( *.pch* ) в текущей компиляции.

## Синтаксис

**/Yu**[имя\_файла]

## 9. Кодировка: определение, назначение, примеры. Кодировка ASCII. Структура кодировки Windows-1251. Отличие ASCII и Windows-1251. Символы времени выполнения: установка кодовой страницы языкового стандарта.

**Кодировка** - таблица, задающая кодировку конечного множества символов алфавита (обычно элементов текста: букв, цифр, знаков препинания). Такая таблица сопоставляет каждому символу последовательность длиной в один или несколько символов другого алфавита (точек и тире в коде Морзе,  нулей и единиц (битов) в компьютере).

Примеры: ASCII, Windows-1251, Кодировки, UTF-8, UTF-16 и UTF-32 набора символов Юникод

**ASCII: American Standard Code for Information Interchange —** американский стандартный код для обмена информацией. ASCII представляет собой 8-битную кодировку для представления десятичных цифр, латинского и национального алфавитов, знаков препинания и управляющих символов. Нижнюю половину кодовой таблицы (0 — 127) занимают символы US-ASCII, а верхнюю (128 — 255) — разные другие нужные символы (CP866, CP1251).

Windows-1251 — набор символов и кодировка, являющаяся стандартной 8-битной кодировкой для русских версий MW.

Кодировка **ASCII**  - кодировка для представления латинского алфавита, десятичных цифр, некоторых знаков препинания, арифметических операций и управляющих символов. **Windows-1251** – для символов русского алфавита.

**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

Дополнительные символы ***времени выполнения*** определяются **setlocale**. По умолчанию, локаль

SetLocale (LC\_ALL, "C")

устанавливает стандартный контекст С.

Во время выполнения можно установить кодовую страницу языкового стандарта, используя вызов setlocale(LC\_CTYPE, "rus")

или

воспользоваться следующими функциями, необходимо включить заголовочный файл <windows.h>:

#include <windows.h> // windows.h содержит прототипы функций SetConsoleOutputCP(1251);//установить кодовую таблицу, на поток ввода SetConsoleCP(1251); //установить кодовую таблицу, на поток

вывода

Директива #pragma позволяет указать целевой языковой стандарт во время компиляции. Это гарантирует, что строки с расширенными символами будут сохраняться в правильном формате.

## 10. Кодировка: определение, назначение, примеры. Кодировка UNICODE: назначение, структура, UCS, UTF. Прямой (LE) и обратный (BE) порядок байт. BOM: определение, назначение, примеры.

**Кодировка** - таблица, задающая кодировку конечного множества символов алфавита (обычно элементов текста: букв, цифр, знаков препинания). Такая таблица сопоставляет каждому символу последовательность длиной в один или несколько символов другого алфавита (точек и тире в коде Морзе,  нулей и единиц (битов) в компьютере).

Примеры: ASCII, Windows-1251, Кодировки, UTF-8, UTF-16 и UTF-32 набора символов Юникод

**UNICODE:** это стандарт кодирования символов, позволяющий представить знаки почти всех письменных [языков](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA), состоит из 2х разделов:

UCS - universal character set (универсальный набор символов);

UTF - Unicode transformation format (семейство кодировок).

Принято обозначение U+xxx, где xxx- число в шестнадцатеричном формате.

**UCS** расположены в 17 плоскостях (0-16), 216 (65 536) символов в каждой плоскости, плоскость 0 – основная (основные символы), 1-14 – дополнительные, 15-16 – для частного использования.

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

Алгоритм кодирования в UTF-8:

* определить количество октетов (октет: 8 битов или 1 байт) – в какой диапазон значений попадает количество значащих символов (7, 11, 16, 21, 26, 31);
* подготовить старшие биты первого октета:
  + 0xxxxxxx для одного октета;
  + 110xxxxx – двух;
  + 1110xxxx - трех и т.д..
  + 10xxxxxx - для остальных октетов;
* заполнить оставшиеся биты (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 с использованием всех возможных диапазонов значений (от 0 до FFFF16).

**LE** (Little endian order, прямой порядок, от младшего к старшему), BE (Big endian order, обратный порядок, от старшего к младшему).

**BOM**: Для определения формата представления Юникода в начало текстового файла записывается сигнатура (обозначение) — символ U+FEFF — маркер последовательности байтов.

|  |  |
| --- | --- |
| **Кодировка** | **Представление (**[**hex**](https://ru.wikipedia.org/wiki/%D0%A8%D0%B5%D1%81%D1%82%D0%BD%D0%B0%D0%B4%D1%86%D0%B0%D1%82%D0%B5%D1%80%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F)**)** |
| [UTF-8](https://ru.wikipedia.org/wiki/UTF-8) | EF BB BF |
| [UTF-16](https://ru.wikipedia.org/wiki/UTF-16) ([BE](https://ru.wikipedia.org/w/index.php?title=Big_Endian&action=edit&redlink=1)) | FE FF |
| [UTF-16](https://ru.wikipedia.org/wiki/UTF-16) ([LE](https://ru.wikipedia.org/w/index.php?title=Little_Endian&action=edit&redlink=1)) | FF FE |
| [UTF-32](https://ru.wikipedia.org/wiki/UTF-32) (BE) | 00 00 FE FF |
| [UTF-32](https://ru.wikipedia.org/wiki/UTF-32) (LE) | FF FE 00 00 |

## 11. Структура языка программирования: алфавит языка программирования, идентификаторы, зарезервированные идентификаторы, литералы, ключевые слова, определения. Правила составления идентификаторов в C++ и других языков программирования. Примеры.

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

Совокупность символов, допускаемых в языке – алфавит языка.

***Базовый набор символов исходного кода:***

1. строчные и прописные буквы латинского и национального алфавитов
2. цифры
3. знаки операций
4. символы подчеркивания \_ и пробела
5. ограничители и разделители
6. специальные символы

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

***Примеры*** правил составления идентификаторов в с++:

* + идентификаторы должны начинаться с буквы или подчеркивания;
  + идентификатор не может совпадать с ключевыми словами С++ или с именами библиотечных функций;
  + идентификаторы могут состоять из любого количества символов, но компилятор **гарантирует**, что будет считать значащими:
    - 31 первых символов идентификаторов, не имеющих внешней связи;
    - не более 6 значащих символов идентификаторов с внешней связью;
  + идентификаторы чувствительны к регистру. Длина идентификатора по стандарту не ограничена.

Идентификатор создается при объявлении переменной, функции, типа и т. п.

**Зарезервированные идентификаторы**: идентификаторы, которые предварительно определены в системе программирования.

**Зарезервированные идентификаторы С++:**

* + все имена с двумя подчеркиваниями считаются зарезервированным;
  + Кроме того: **is**xxxx, **mem**xxxx, **str**xxxx, **t**oxxxx, **wcs**xxxx, **Eцифра**xxxx, **LC\_X**xxx, **SIGX**xxx, **SIG\_X**xxxx.

**Литерал: элемент** программы, который непосредственно представляет значение. В C++ существует четыре типа литералов:

* целочисленный литерал,
* вещественный литерал,
* символьный литерал,
* строковый литерал.

|  |  |  |  |
| --- | --- | --- | --- |
| Управляющие символьные литералы:  \0 \x00 null пустая литера | | | |
| \a | \x07 | bel | сигнал |
| \b | \x08 | bs | возврат на шаг |
| \f | \x0C | ff | перевод страницы |
| \n | \x0A | lf | перевод строки |
| \r | \x0D | cr | возврат каретки |
| \t | \x09 | ht | горизонтальная табуляция |
| \v | \x0B | vt | вертикальная табуляция |
| \\ | \x5C | \ | обратная косая черта |
| \' | \x27 | ' |  |
| \" | \x22 | " |  |
| \? | \x3F | ? |  |

**Ключевые слова**: последовательности символов алфавита языка, имеющие специальное назначение. Ключевые слова зарезервированы компилятором для обозначения типов переменных, класса хранения, элементов операторов и т.д.

**С++**

|  |  |  |  |
| --- | --- | --- | --- |
| [break](https://docs.microsoft.com/en-us/cpp/cpp/break-statement-cpp?view=vs-2017) | [case](https://docs.microsoft.com/en-us/cpp/cpp/switch-statement-cpp?view=vs-2017) | [catch](https://docs.microsoft.com/en-us/cpp/cpp/try-throw-and-catch-statements-cpp?view=vs-2017) | [char](https://docs.microsoft.com/en-us/cpp/cpp/fundamental-types-cpp?view=vs-2017) |
| [char16\_t](https://docs.microsoft.com/en-us/cpp/cpp/char-wchar-t-char16-t-char32-t?view=vs-2017) | [char32\_t](https://docs.microsoft.com/en-us/cpp/cpp/char-wchar-t-char16-t-char32-t?view=vs-2017) | [class](https://docs.microsoft.com/en-us/cpp/cpp/class-cpp?view=vs-2017) | [const](https://docs.microsoft.com/en-us/cpp/cpp/const-cpp?view=vs-2017) |
| [false](https://docs.microsoft.com/en-us/cpp/cpp/false-cpp?view=vs-2017) | [finally](https://docs.microsoft.com/en-us/cpp/dotnet/finally?view=vs-2017) | [float](https://docs.microsoft.com/en-us/cpp/cpp/fundamental-types-cpp?view=vs-2017) | [for](https://docs.microsoft.com/en-us/cpp/cpp/for-statement-cpp?view=vs-2017) |
| [inline](https://docs.microsoft.com/en-us/cpp/cpp/inline-functions-cpp?view=vs-2017) | return | **if** | struct |
| [cdecl](https://docs.microsoft.com/en-us/cpp/cpp/cdecl?view=vs-2017) | [int16](https://docs.microsoft.com/en-us/cpp/cpp/int8-int16-int32-int64?view=vs-2017) 4 | [int32](https://docs.microsoft.com/en-us/cpp/cpp/int8-int16-int32-int64?view=vs-2017) 4 | [int64](https://docs.microsoft.com/en-us/cpp/cpp/int8-int16-int32-int64?view=vs-2017) |