1. Утверждаю
2. Директор института СПИНТех
3. НИУ МИЭТ
4. Проф. \_\_\_\_\_\_\_\_\_\_/Гагарина Л.Г./
5. «\_\_\_» \_\_\_\_\_\_\_\_\_ 2021 г.
6. Пояснительная записка

**Направление подготовки – 09.03.04**

1. **Квалификация – бакалавр**

Руководитель выпускной работы:

К.т.н., профессор \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Касимов Р.А./

1. Исполнитель:
2. Студент гр. ПИН-41 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Федотов А.А./

Москва 2021

Содержание

[Перечень используемых сокращений 3](#_Toc72592248)

[Введение 4](#_Toc72592249)

[1. Исследовательский раздел 5](#_Toc72592250)

[1.1 Протоколы синхронизации времени 5](#_Toc72592251)

[1.2 Виды сетевых устройств 8](#_Toc72592252)

[1.3 Обзор аналогичных программных решений 12](#_Toc72592253)

[1.4 Требования к ПМ НКСС. 19](#_Toc72592254)

[1.5 Постановка цели и задач 20](#_Toc72592255)

[Предполагаемый алгоритм решения 20](#_Toc72592256)

[Выводы по разделу 20](#_Toc72592257)

[2. Конструкторский раздел 21](#_Toc72592258)

[2.1 Выбор языка 21](#_Toc72592259)

[2.3 Схема алгоритма 37](#_Toc72592260)

[Организация связи с другими модулями. 38](#_Toc72592261)

[Выводы по разделу 39](#_Toc72592262)

[3. Технологический раздел 40](#_Toc72592263)

[3.1 Методы тестирования 42](#_Toc72592264)

[3.2 Выбор средств для тестирования ПМ НКСС 45](#_Toc72592265)

[3.3 Прототипирование 46](#_Toc72592266)

[3.4 Отладка по точкам останова 47](#_Toc72592267)

[3.5 Отладка с помощью трассировки 48](#_Toc72592268)

[3.6 Отладка ПМ НКСС 50](#_Toc72592269)

[3.7 Модульное тестирование 51](#_Toc72592270)

[Выводы по разделу 54](#_Toc72592271)

# Перечень используемых сокращений

ПМ – Программный модуль

НКСС – настройка конфигурации сетевых сервисов

ПАК – программно-аппаратный комплекс

NTP – Network Time Protocol

ЯП – язык программирования

IDE (англ. Integrated Development Environment) - интегрированная среда разработки

JVM - виртуальная машина Java

ПО – программное обеспечение

SNTP (англ. Simple Network Time Protocol) – упрощенный протокол NTP

OSI (англ. Open Systems Interconnection model) — сетевая модель стека сетвых протоколов OSI/ISO.

RFC - документ из серии пронумерованных информационных документов Интернета, содержащих технические спецификации и стандарты, широко применяемые во всемирной сети.

VS (Visual Studio) – среда разработки ПО

UDP

UTC - это всемирное координированное время, стандарт времени, принятый на Земле.

CLR

BCL

# Введение

Информационные технологии все больше и больше проникают в жизнь людей. С их развитием появилась необходимость во взаимодействии компьютеров между собой. Для того, чтобы это взаимодействие обеспечить, создаются и постоянно улучшаются различные сетевые сервисы, которые позволяют решать те или иные задачи. Одной из таких задач является синхронизация времени сетевых устройств. Синхронизация времени необходима для согласования работы устройств и приложений, осуществляющих обработку данных в режиме реального времени. Кроме того, она необходима в системах управления для протоколирования каких-либо событий и своевременного реагирования на них. Для контроля синхронизации используется протокол NTP (Network Time Protocol – протокол сетевого времени), однако, чтобы правила протокола соблюдались, нужно правильно настроить конфигурацию устройства.

Целью данной работы является создание программного модуля, реализующего настройку конфигурации NTP на сетевых устройствах.

Выполнение выпускной квалификационной работы проходило в компании ООО «С-Терра СиЭсПи», специализирующейся на разработке и производстве средств для обеспечения сетевой информационной безопасности, а также ПО для реализации этих средств.

Разрабатываемый модуль ПМ НКСС имеет высокую практическую значимость при решении задач по настройке синхронизации сетевого времени на устройствах, поддерживающих протокол NTP.

Пояснительная записка состоит из введения, исследовательского, конструкторского, технического разделов, заключения и списка литературы.

В исследовательском разделе производится анализ предметной области настройки конфигурации сетевых устройств по протоколу NTP, рассматривается актуальность выбранной темы, производится сравнение существующих решений и выявляются их недостатки.

В конструкторском разделе выбирается язык и среда программирования, определяются необходимые библиотеки, производится разработка алгоритма программного модуля.

# Исследовательский раздел

Все сетевые устройства имеют внутренние часы. Они инициализируются при загрузке системы, затем время уже поддерживается с помощью регулярных прерываний от таймера, так что они работают даже при выключенном устройстве. Эти внутренние часы отслеживают как время, так и дату. Важно следить за точностью часов, иначе могут возникнуть различные проблемы с корреляцией логов, ПО и т. Д.

Рассмотрим некоторые протоколы, которые используются для синхронизации времени на разных устройствах.

# Протоколы синхронизации времени

Daytime и Time

Впервые на компьютерах появились и пока используются протоколы DAYTIME (описан в RFC 867) и TIME (описан в RFC 868). Первый из них предназначен для передачи и показа даты и времени в понятной человеку форме, второй – в форме, понятной компьютеру.

Формат ответа DAYTIME строго не регламентирован и не предназначен для машинной обработки - предполагается только, что человек, прочитавший полученную строку, поймет текущее время и дату.

Протокол TIME, предназначен для обмена времени между машинами. На подключившийся к TIME-серверу компьютер приходит UDP-пакет, содержащий единственное 32-битное беззнаковое число, соответствующе числу прошедших с 1 января 1900 г. секунд по UTC. Такое число переполняется через 136 лет, а значит, этот протокол способен функционировать только до 2036 года.

NTP – протокол

Ни DAYTIME, ни TIME не могут обеспечить необходимую точность синхронизации времени. В связи с этим, в 1985 г. Дэвид Л. Миллс (David L. Mills) из университета Дэлавера разработал сетевой протокол синхронизации времени NTP, точнее его первоначальную, позже названную нулевой (NTPv0) версию, описанную в RFC 958.

Протокол NTP использует алгоритм Марзулло (предложен Кейтом Марзулло (Keith Marzullo) из Университета Калифорнии, Сан-Диего). В версии 4 (NTPv4) способен достигать точности 10 мс при работе через Интернет, и до 0,2 мс внутри локальных сетей, включая такую особенность, как учёт времени передачи.

Как известно, этот протокол используется для синхронизации времени между устройствами. Для своей работы он использует протокол UDP, при этом учитывает время передачи. Система протокола очень устойчива к изменениям задержки (в данном случае имеется в виду время для прохождения пакета данных от одной точки к другой).

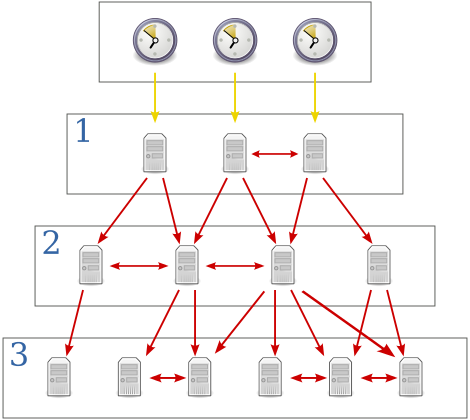


Рисунок Схема работы NTP протокола.

NTP-серверы работают в иерархической сети (рисунок 1). Каждый уровень этой сети называют ярусом (stratum). На самом верхнем (нулевом, stratum0) уровне находятся эталонные часы. В качестве таких часов используются сигнал GPS (Global Positioning System) или службы ACTS (Automated Computer Time Service). Сервера первого яруса получают данные от эталонных часов. Далее сервера второго яруса синхронизируются с серверами первого яруса, сервера третьего яруса синхронизируются с серверами второго яруса и так далее по аналогии. Всего может быть до 15 ярусов.

NTP-серверы и NTP-клиенты получают данные о времени от серверов яруса 1, хотя на практике NTP-клиентам лучше не делать этого, поскольку тысячи индивидуальных клиентских запросов окажутся слишком большой нагрузкой для серверов яруса 1. Лучше настроить локальный NTP-сервер, который клиенты будут использовать для получения информации о времени.

Стоит отметить, что протокол не устанавливает время на устройстве в чистом виде. Он корректирует время с использованием временного смещения – разницы между временем на NTP-сервере и на внутренних часах.

Протокол NTP постоянно совершенствовался: NTPv1 (1988 г, RFC 1059), NTPv2 (1989 г., RFC1119), NTPv3 (1992 г., RFC1305), NTPv4 (1996 г., RFC2030).

SNTP протокол

Существует упрощенная версия протокола NTP - SNTP (Simple Network Time Protocol). Он был реализован для того, чтобы синхронизировать время конечным клиентом.

Поскольку все преимущества протокола NTP проявляются именно в сети серверов, для получения показаний конечным пользователем NTP излишне сложен. Чтобы решить задачу синхронизации времени конечными компьютерами и серверами был предложен протокол SNTP (SNTPv3: 1992 г., RFC1361 и 1995 г., RFC1769; SNTPv4 включён как вложенный в NTPv4).

SNTP — это не новый протокол, а способ использования NTP-пакетов и NTP-серверов в приложениях, где не требуется высокая точность времени, либо такая точность недостижима. В этом случае клиент использует только часть информации UDP-пакета NTP-сервера. SNTP-клиент может работать с любыми версиями NTP-серверов, и кроме них - с особыми SNTP-серверами, которые в откликах заполняют только необходимые данные UDP-пакета.

Таким образом, облегченный протокол SNTP образует не сеть синхронизирующихся между собой NTP-серверов, а пары "клиент-сервер". Любой NTP-сервер является одновременно SNTP-сервером. Клиент, который не передаёт полученное время дальше, может работать как NTP- или SNTP-клиент, в зависимости от условий. Для SNTP, как и для NTP, зарезервирован 123-й UDP-порт.

В большинстве сетевых устройств используется протокол NTP. Разберем некоторые примеры сетевых устройств, которые при своей работе используют данный протокол.

* 1. Виды сетевых устройств

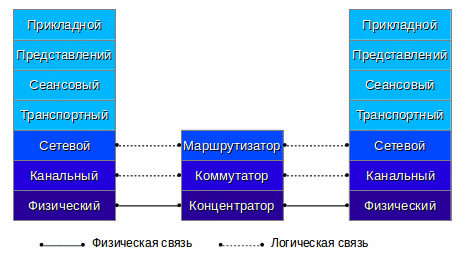


Рисунок модель OSI

Прежде чем разбирать виды сетевых устройств, нужно рассказать про сетевую модель OSI. Данная модель определяет многоуровневое взаимодействие между узлами сети, где каждый уровень представляет набор специфичных функций. Нижние уровни: физический, канальный и сетевой — регламентируют процесс передачи данных как таковой. Сетевые интерфейсы оконечных узлов представляют эти функции, но этого недостаточно, чтобы обеспечить связь между произвольными узлами в локальной сети и, тем более, в Интернет. Причина проста — невозможно установить непосредственные физические связи между всеми узлами. Для снятия этого и прочих ограничений используется дополнительное сетевое оборудование. Основными его типами являются:

* Концентраторы
* Коммутаторы
* Маршрутизаторы

Концентратор

Концентратор (также хаб от англ. hub — центр) – сетевое устройство, работающее на первом уровне. Основные его задачи связаны с ретрансляцией поступающих данных на все остальные подключенные порты. Никаких действий по обработке данных не производится. Благодаря этому, сеть Ethernet построенная на концентраторах имеет все недостатки общей шины кроме одного: вывод из строя некоторого узла не приводит к остановке работы всей сети. На сегодняшний день концентраторы почти не используются, уступив место более совершенным устройствам – коммутаторам. Единственное преимущество концентратора — это его низкая стоимость. Оно было актуально лишь в первые годы развития сетей Ethernet. По мере того, как совершенствовались и становились дешевле электронные микропроцессорные компоненты, данное преимущество концентратора полностью нивелировалось, так как стоимость вычислительной части коммутаторов и маршрутизаторов составляет лишь малую долю на фоне стоимости разъёмов, разделительных трансформаторов, корпуса и блока питания, общих для концентратора и коммутатора.

Недостатки концентратора являются логическим продолжением недостатков, присущих топологии «общая шина», а именно — это снижение пропускной способности сети по мере увеличения числа узлов. Кроме того, поскольку на физическом уровне узлы не изолированы друг от друга, все они будут работать со скоростью передачи данных самого худшего узла. Например, если в сети присутствуют узлы со скоростью 100 Мбит/с и всего один узел со скоростью 10 Мбит/с, то все узлы будут работать на скорости 10 Мбит/с, даже если узел 10 Мбит/с вообще не проявляет никакой информационной активности. Ещё одним недостатком является вещание сетевого трафика во все порты, что снижает уровень сетевой безопасности и даёт возможность подключения и успешного использования снифферов (считыватели траффика).



Рисунок сетевой концентратор

Коммутатор

Сетевой коммутатор (свитч) – устройство, которое предназначено для соединения нескольких узлов компьютерной сети в пределах одного или нескольких сегментов сети (рисунок 4). Коммутатор хранит в памяти таблицу коммутации, в которой указывается соответствие МАС-адреса узла порту коммутатора. При включении эта таблица пуста, и он работает в режиме обучения. В таком режиме поступающие на какой-либо порт данные передаются на все остальные порты, а МАС-адрес порта-отправителя заносится в таблицу. Если же МАС-адрес хоста-получателя имеется в таблице, данные передаются только получателю. Таким образом, со временем проходящий трафик локализируется.

Существует три способа коммутации. Каждый из них — это комбинация таких параметров, как время ожидания и надёжность передачи.

* С промежуточным хранением (Store and Forward). Коммутатор читает всю информацию в кадре, проверяет его на отсутствие ошибок, выбирает порт коммутации и после этого посылает в него кадр.
* Сквозной (cut-through). Коммутатор считывает в кадре только адрес назначения и после выполняет коммутацию. Этот режим уменьшает задержки при передаче, но в нём нет метода обнаружения ошибок.
* Бесфрагментный (fragment-free) или гибридный . Этот режим является модификацией сквозного режима, который частично решает проблему коллизий. В теории поврежденные кадры (обычно из-за столкновений) часто короче минимального допустимого размера кадра Ethernet, равного 64 байтам. Поэтому в этом режиме коммутатор отбрасывает кадры длиной меньше 64 байт, а все остальные после прочтения первых 64 байт в сквозном режиме передаёт дальше



Рисунок сетевой коммутатор

Маршрутизатор

Маршрутизатор (роутер) – устройство, работающее на третьем уровне и выполняющее функции перенаправления трафика между сетями. Маршрутизатор способен связывать разнородные сети различных архитектур. Для принятия решения о пересылке пакетов данных, используется информация о топологии сети и определенные правила, которые задаются администратором. Обычно маршрутизатор использует адрес получателя и по таблице маршрутизации определяет путь, по которому следует передать данные дальше. Если в таблице такого маршрута нет – пакет отбрасывается.

Существуют и другие способы определения маршрута пересылки пакетов, когда, например, используется адрес отправителя, используемые протоколы верхних уровней и другая информация, содержащаяся в заголовках пакетов сетевого уровня. Нередко маршрутизаторы могут осуществлять трансляцию адресов отправителя и получателя, фильтрацию транзитного потока данных на основе определённых правил с целью ограничения доступа, шифрование/расшифровывание передаваемых данных и т. д.



Рисунок маршрутизатор

* 1. Обзор аналогичных программных решений

В ходе предварительных исследований был проведен анализ существующих решений с функционалом, требуемым от разрабатываемого программного модуля. При анализе учитывался не только необходимый функционал: настройка параметров для синхронизации времени на устройствах по протоколу NTP, но и смежные возможности, облегчающие использование решения; возможности компании разработчика по актуализации функционала, и его расширению. Характеристики рассмотренных решений в сравнении с ПМ НКСС представлены в табл. 1.

Таблица

Сравнение аналогов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Аналоги  Критерии | MikroTik RouterOS | Quagga | Junos OS | Cisco IOS | C-Терра Шлюз |
| Возможность настройки NTP | Есть | Нет | Есть | Есть | Есть |
| Способ настройки NTP конфигурации | С помощью конфигурационных файлов и команд | Нет | С помощью команд | С помощью команд | С помощью команд |
| Возможность сохранения логов NTP в отдельный файл | Нет | Нет | Нет | Нет | Да |
| Необходимость загрузки дополнительных файлов | Есть | Нет | Нет | Нет | Нет |
| Наличие пользовательского интерфейса | Графический интерфейс | Консоль | Графический интерфейс | Графический интерфейс | Консоль |

MikroTik RouterOS

MikroTik — это производитель сетевых устройств из Латвии. Компания занимается разработкой и продажей как проводного, так и беспроводного сетевого оборудования, в том числе маршрутизаторы, сетевые коммутаторы (коммутаторы), точки доступа, а также программное обеспечение, куда входят операционные системы и вспомогательное ПО. Эта компания основана в 1996 году с для продажи оборудования на развивающихся рынках.

Одним из продуктов данной компании является RouterOS, сетевая операционная система на базе Linux. Эта система предназначена для установки на маршрутизаторы MikroTik RouterBoard. Также есть возможность установить эту систему на ПК, превратив его в маршрутизатор с функциями брандмауэра, VPN-сервера / клиента, QoS, точки доступа и других. Система также может служить в качестве адаптивного портала на основе системы беспроводного доступа. Также существует специальная версия RouterOS, называемая Cloud Hosted Router, для облачных виртуальных машин

Описываемая операционная система имеет несколько уровней лицензий. С каждым уровнем возрастает число функций. Кроме того, существует ПО под названием Winbox. Это ПО предоставляет пользователю графический интерфейс (GUI) для настройки ОС. Доступ к устройствам, находящимся под управлением RouterOS возможен также через веб-интерфейс, FTP, Telnet, и SSH. Существует интерфейс программирования приложений, который позволяет создавать специализированные приложения для управления и мониторинга.

RouterOS может поддерживать множество сервисов и протоколов, которые могут использовать средние или крупные провайдеры, такие как OSPF, BGP, VPLS/MPLS. ОС является достаточно гибкой системой, и хорошо поддерживается компанией, как в рамках форума и предоставления различных Wiki-материалов, так и специализированных примеров конфигураций.

Данная система обеспечивает поддержку почти всех сетевых интерфейсов на ядре Linux. Из беспроводных чипсетов поддерживаются решения на основе Atheros и Prism (по состоянию RouterOS версии 3.x). Mikrotik также работает над модернизацией программного обеспечения, которая обеспечит полную совместимость устройств и ПО Mikrotik с набирающими популярность сетевыми технологиями, такими как IPv6.

RouterOS предоставляет администратору графический интерфейс для более наглядной и простой настройки файервола, маршрутизации и управления QoS. Также, в интерфейсе WinBox практически полностью реализована функциональность Linux-утилит iptables, iproute2, управление трафиком и QoS на основе алгоритма HTB.

Quagga

Quagga — пакет свободного программного обеспечения, поддерживающий протоколы динамической маршрутизации IP. Компьютер с установленным и сконфигурированным пакетом Quagga становится способен использовать любые из перечисленных ниже протоколов динамической маршрутизации:

* Routing Information Protocol (RIP): v1-3;
* Open Shortest Path First (OSPF): v2-3;
* Border Gateway Protocol (BGP): v4;
* Intermediate System to Intermediate System (IS-IS);
* Protocol Independent Multicast (PIM, только PIM-SSM).

Пакет Quagga может быть установлен на UNIX-подобные операционные системы. Quagga — это усовершенствованная версия GNU Zebra, компьютерной программы, развитие которой остановилось в 2005 году.

Свое название продукт получил от животного. Квагга (Quaga) — это подвид зебры, обитавший в южной Африке. В отличие от самого животного, истреблённого в конце XIX‐ого века, проект Quagga пережил ныне умерший проект GNU Zebra. Последняя стабильная версия Zebra (0.95a) датирована 2005-09-08, большинство BGP маршрутизаторов, которые использовали GNU Zebra, перешли на Quagga.

Quagga состоит из базового ядра (core daemon) zebra, выполняющего роль промежуточного уровня абстракции (abstraction layer) ядра ОС, и предоставляющего Zserv API клиентам по протоколу TCP. Клиентами Zserv выступают службы (демоны):

* ospfd (протокол OSPFv2);
* ripd (протокол RIP v1, V2);
* ospf6d (протокол OSPFv3 IPv6);
* ripngd (протокол RIPng IPv6);
* bgpd (протокол BGPv4+, включая поддержку multicast и IPv6));
* isisd (протокол IS-IS);
* pimd (протокол PIM, пока только PIM-SSM).

Библиотека Quagga существенно облегчает разработку дополнительных модулей, позволяя всем её службам использовать унифицированный способ конфигурации и управления.

**JUNOS**

JUNOS — это операционная система, которая используется на оборудовании компании Juniper Networks. Была создана на основе свободной ОС FreeBSD. Главным конкурентом junos считается операционная система IOS-XR компании Cisco Systems.

В JUNOS есть возможность установки дополнительного ПО, которое распространяется в виде пакетов, подписанных соответствующим сертификатом компании. Система управления пакетами JUNOS также унаследована от FreeBSD.

Пользовательское окружение представляет из себя полноценную рабочую среду, включающую набор классических для FreeBSD утилит. Однако, внесение каких-либо изменений в конфигурацию допускается только с помощью специальной утилиты — «cli». Этой же утилитой осуществляется управление всеми ASIC.

Серии оборудования Juniper Networks, работающие под управлением JUNOS:

* сервисные шлюзы — SRX
* сервисные маршрутизаторы — J
* маршрутизаторы — M, MX, T
* коммутаторы — EX
* коммутаторы для ЦОД – QFX

Командный интерфейс ОС позволяет не только выполнять команды, но и вводить конфигурацию. Изменения в конфигурации не применяются до того, как будет выполнена команда commit. В ПО предусмотрена возможность автоматического отката конфигурации на предыдущую, если она не была подтверждена в течение заданного времени. Поддерживается история версий (вплоть до 50 версий), к которым может производиться возврат. Как и большинство других интерфейсов командной строки маршрутизаторов, junos поддерживает автодополнение по уникальной комбинации первых символов команды (например, sh int fe-1/1/1 ex будет расшифрованно как show interface fe-1/1/1 extensive). Автодополнение происходит не при интерпретации команды, а при нажатии пробела. Поддерживается контекстная справка (вызов — знак вопроса).

Конфигурация представляет собой директивы конфигурирования той или иной подсистемы. Эти директивы могут содержать в себе вложенные элементы, которые описывают настройку отдельных компонент. Например, конфигурация интерфейса ethernet может иметь вложенные настройки для отдельных подсетей, которые, в свою очередь, могут иметь вложенные настройки для различных протоколов (например, ip4 и ip6).

Оконечные узлы конфигурации (те, что не содержат вложенных элементов) заканчиваются точкой с запятой, содержащие вложенные элементы задают их с помощью фигурных скобок (точка с запятой в этом случае не ставится). Конфигурирование осуществляться либо вводом готового текстового блока, либо использованием сокращённых директив с помощью команды set.

Cisco IOS

Cisco IOS (от англ. Internetwork Operating System — Межсетевая Операционная Система) — программное обеспечение, которое используется в маршрутизаторах и сетевых коммутаторах Cisco. Cisco IOS является операционной системой, которая выполняет функции сетевой организации, маршрутизации, коммутации и передачи данных.

У Cisco IOS есть довольно специфичный интерфейс командной строки (command line interface, CLI), который был скопирован и все еще копируется многими другими сетевыми продуктами. Интерфейс предлагает набор многословных команд, в зависимости от выбранного режима и уровня привилегий пользователя. Global configuration mode дает возможность изменять настройки системы и сетевых интерфейсов.

Всем командам интерфейса приписывается определённый уровень привилегий от 0 до 15, к которым могут обратиться только те пользователи, у которых соответствующий уровень привилегий. С помощью командного интерфейса можно определить доступные команды для каждого уровня привилегий.

Существуют разные компоновки IOS отличающиеся функционалом, так называемые feature sets :

* IP Base — начальный уровень функциональности, включается во все другие. Обеспечивает базовую маршрутизацию.
* IP Services (для L3 свитчей) — протоколы динамической маршрутизации, NAT, IP SLA.
* Advanced IP Services — добавляется поддержка IPv6.
* IP Voice — добавляет функциональность VoIP и VoFR.
* Advanced Security — добавляется IOS/Firewall, IDS, SCTP, SSH и IPSec (DES, 3DES и AES).
* Service Provider Services — добавляется Netflow, SSH, BGP, ATM и VoATM.
* Enterprise Base — добавляется поддержка L3-протоколов (IPX и AppleTalk), а также DLSw+, STUN/BSTUN и RSRB.

**С-Терра Шлюз**

ПАК «С-Терра Шлюз» является программно-аппаратным средством защиты сетей, подсетей, офисов и самого шлюза от несанкционированного доступа.

ПАК «С-Терра Шлюз» используется для защиты от несанкционированного доступа к информации ограниченного доступа, не содержащей сведений, составляющих государственную тайну. ПАК «С-Терра Шлюз» может применяться:

* в значимых объектах критической информационной инфраструктуры (КИИ) до 1 категории включительно;
* в государственных информационных системах (ГИС) до 1 класса защищенности включительно;
* в автоматизированных системах управления производственными и технологическими процессами (АСУТП) до 1 класса защищенности включительно;
* в информационных системах персональных данных (ИСПДн), обеспечивающих 1, 2, 3 и 4 уровни защищенности персональных данных;
* в информационных системах общего пользования (ИСОП) II класса.

ПАК «С-Терра Шлюз» предназначен для работы на аппаратных платформах в архитектуре Intel (x86-64 совместимых) универсального назначения, отвечающих следующим минимальным требованиям: имеющих от 1 процессора, от 2 Гб ОЗУ, от 3 сетевых интерфейсов, подключаемых к внешним сетям:

* Аквариус T30S100DC, T30S001DC, T40S102DF-B;
* LN-S, LN-Si, LN-M, LN-L, LN-XL;
* АТБ-АТОМ-1.

ПАК «С-Терра Шлюз» работает под управлением операционной системы Debian Linux 9 с установленными последними обновлениями безопасности.

* 1. Требования к ПМ НКСС.

ПМ НКСС должен обеспечивать выполнение следующих функций:

* Правильное распознавание введенных команд
* При вводе неверной команды должно выводиться сообщение об ошибке с указанием места ошибки
* Должна быть возможность настройки конфигурации в качестве NTP-клиента
* Должна быть возможность настройки конфигурации в качестве NTP-сервера
* Должна быть возможность указания NTP-серверов, с которыми проводится синхронизация
* Должна быть возможность с просмотра статуса текущих настроек с помощью определенных команд
* Должна быть предусмотрена возможность сохранения логов в отдельный лог-файл
* Все возвращаемые данные должны выводиться обратно на командную строку
  1. Постановка цели и задач

Цель разработки:

Создать программный модуль, позволяющий упростить настройку конфигурации NTP сервисов на сетевых устройствах.

**Задачи:**

* Исследование предметной области
* Сравнительный анализ существующих решений
* Выбор платформы для реализации модуля
* Выбор языка и среды разработки
* Разработка схемы данных ПМ НКСС
* Разработка схемы алгоритма ПМ НКСС
* Программная реализация ПМ НКСС
* Тестирование и отладка модуля
* Разработка руководства оператора

Предполагаемый алгоритм решения

ПМ получает команду и необходимые данные из консоли. На основе введенной команды выполняются действия по изменению настроек конфигурации или формировании сообщения о текущих настройках NTP.

Если в настройках указан файл для логирования, то в этот файл сохраняются логи работы NTP-сервиса.

# Выводы по разделу

В исследовательском разделе была обоснована актуальность разработки ПМ НКСС. Исследована предметная область и проведен сравнительный анализ решений для настройки конфигурации NTP на сетевых устройствах.

# Конструкторский раздел

* 1. Выбор языка

В ходе исследовательской работы был проведен сравнительный анализ языков программирования, результаты которого приведены ниже в таблице.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Язык  Критерий | С++ | Java | Python | C# | Objective C |
| Скорость работы | Высокая | Высокая, но ниже чем можно добиться на С++ | Высокая, но ниже чем можно добиться на С++ | Высокая, но ниже чем можно добиться на С++ | Высокая |
| Знание языка, опыт работы с ним | Есть | Есть | Нет | Нет | Нет |
| Синтаксис (удобство разработки) | Не очень удобный | Удобный | Не удобный | Удобный | Не удобный |
| Использование языка в других проектах кампании | Используется | Используется | Не используется | Не используется | Не используется |
| Простота соединения со смежными ПМ | Просто, т.к смежные модули написаны на этом языке | Сложно | Сложно | Сложно | Сложно |

C++

C++ (читается как си-плюс-плюс) — компилируемый, язык программирования общего назначения со статической типизацией.

Может поддерживать различные парадигмы программирования, такие как:

* процедурное программирование
* объектно-ориентированное программирование,
* обобщённое программирование.

Язык имеет довольно богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы: ввод-вывод, регулярные выражения, поддержка многопоточности и многие другие возможности.

Язык сочетает в себе свойства как высокоуровневых, так и низкоуровневых языков. По сравнению с его предшественником (языком C), наибольшее внимание уделено объектно-ориентированному и обобщённому программированию.

C++ довольно широко используется при разработке ПО и является одним из самых популярных языков программирования. Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также разработка игр. Существует множество различных реализаций языка, как бесплатных, так и коммерческих для различных платформ. Например, на платформе x86 это GCC, Visual C++, Intel C++ Compiler, Embarcadero (Borland) C++ Builder и другие. C++ оказал огромное влияние на другие языки программирования, в первую очередь на Java и C#.

Синтаксис C++ был унаследован от языка C. Одним из принципов его разработки было сохранение совместимости с C. Тем не менее C++ не совсем является надмножеством C. Множество программ, которые могут одинаково успешно транслироваться как компиляторами C, так и компиляторами C++, довольно велико, но не включает все возможные программы на C.

C#

C# (произносится как си шарп) — объектно-ориентированный язык программирования. Был разработан в 1998—2001 годах группой инженеров компании Microsoft. Руководили разработкой Андерс Хейлсберг и Скотт Вильтаумот. С# разрабатывался как язык для приложений платформы Microsoft .NET Framework. Впоследствии он был стандартизирован как ECMA-334 и ISO/IEC 23270.

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, переменные, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML и статическую типизацию.

Язык многое перенял от своих предшственников (языков C++, Delphi, Модула, Smalltalk и, в особенности, Java). Опираясь на практику их использования, C# исключает некоторые модели, которые зарекомендовали себя как проблематичные при разработке ПО.

C# разрабатывался как язык программирования прикладного уровня для CLR и, как таковой, зависит, прежде всего, от возможностей самой CLR. Это касается, прежде всего, системы типов C#, которая отражает BCL. Присутствие или отсутствие тех или иных выразительных особенностей языка диктуется тем, может ли конкретная языковая особенность быть транслирована в соответствующие конструкции CLR.

С развитием CLR от версии 1.1 к версии 2.0 сильно обогатился и сам C#. Подобного взаимодействия следует ожидать и в дальнейшем, однако, эта закономерность была нарушена с выходом C# версии 3.0, представляющего собой расширения языка, не опирающиеся на расширения платформы .NET. CLR предоставляет C#, как и всем другим .NET-ориентированным языкам, многие возможности, которых лишены «классические» языки программирования. Например, сборка мусора не реализована в самом C#, а производится CLR для программ, написанных на C# точно так же, как это делается для программ на VB.NET, J# и др.

Objective-C

Objective-C — компилируемый объектно-ориентированный язык программирования, который используется корпорацией Apple. Он был построенна основе языка Си и парадигм Smalltalk. В частности, объектная модель построена в стиле Smalltalk — то есть объектам посылаются сообщения. Язык Objective-C является надмножеством для языка Си, поэтому код, написанный на Си полностью понятен компилятору Objective-C.

В начале 1980-х годов было довольно популярно структурное программирование. Оно позволяло разделить алгоритм на небольшие блоки. Однако, по мере роста сложности задач, структурное программирование приводило к снижению качества кода и удобства разработки. Приходилось писать всё больше функций, которые очень редко могли использоваться в других программах.

Многие программисты увидели в объектно-ориентированном программировании потенциальное решение возникшей проблемы. С одной стороны, Smalltalk использовали почти все более-менее сложные системы. С другой — использование виртуальных машин повышало требования к ресурсам.

Objective-C был создан Брэдом Коксом в начале 1980-х в его компании Stepstone. Он пытался решить проблему повторного использования кода.

Целью Брэда Кокса было создание такого языка, который бы поддерживал концепцию software IC, подразумевающей возможность собирать программы из готовых компонентов (объектов), подобно тому, как сложные электронные устройства могут быть собраны из набора готовых интегральных микросхем.

При этом разрабатываемый язык должен был быть простым и основанным на языке С, чтобы облегчить переход разработчиков на него. Одной из целей было также создание модели, в которой сами классы являются полноценными объектами, поддерживалась бы интроспекция и динамическая обработка сообщений.

Objective-C является расширением С: любая программа на С является программой на Objective-C. Одной из отличительных черт Objective-C является динамичность: решения, обычно принимаемые на этапе компиляции, здесь откладываются до этапа выполнения.

Objective-C — message-oriented-язык, в то время как C++ — function-oriented: в Objective-C вызовы метода интерпретируются не как вызов функции (хотя к этому обычно все сводится), а как посылка сообщения (с именем и аргументами) объекту, подобно тому, как это происходит в Smalltalk.

Есть возможность любому объекту послать любое сообщение. Объект может вместо обработки сообщения переслать его другому объекту для обработки (делегирование), в частности, так можно реализовать распределённые (то есть находящиеся в различных адресных пространствах и даже на разных компьютерах) объекты. Привязка сообщения к соответствующей функции происходит на этапе выполнения.

Язык Objective-C может работать с метаинформацией. На этапе выполнения есть возможность узнать класс объекта, список его методов (с типами передаваемых аргументов) и instance-переменных, проверить, является ли класс потомком заданного и поддерживает ли он заданный протокол и т. п.

В языке также есть поддержка протоколов (понятия интерфейса объекта и протокола четко разделены). Поддерживается наследование (не множественное); для протоколов поддерживается множественное наследование. Объект может быть унаследован от другого объекта и сразу нескольких протоколов (хотя это скорее не наследование протокола, а его поддержка). На данный момент язык Objective-C поддерживается компиляторами Clang и GCC (под управлением Windows используется в составе MinGW или cygwin).

Некоторые функции языка перенесены в runtime-библиотеку и сильно зависят от неё. Вместе с компилятором gcc поставляется минимальный вариант такой библиотеки. Также можно свободно скачать runtime-библиотеку компании Apple: Apple’s Objective-C runtime. Эти две runtime-библиотеки похожи (основные отличия в именах методов).

Java

Java (читается как джава) — объектно-ориентированный язык программирования общего назначения, со строгой типизацией. Был разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle). Разработка ведётся сообществом, организованным через Java Community Process. Язык и основные реализующие его технологии распространяются по лицензии. Права на торговую марку принадлежат корпорации Oracle.

Приложения, разработанные на Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре, для которой существует реализация виртуальной Java-машины (JVM – Java Virtual Machine). Дата официального выпуска — 23 мая 1995 года. Стабильно занимает высокие места в рейтингах популярности языков программирования (2-е место в рейтингах IEEE Spectrum (2020) и TIOBE (2021)).

Изначально этот язык назывался Oak («Дуб»). Он разрабатывался Джеймсом Гослингом для разработки ПО для бытовых электронных устройств. Из-за того, что язык с таким названием уже существовал, Oak был переименован в Java. Язык получил название в честь марки кофе Java, которая, в свою очередь, получила наименование одноимённого острова (Ява), поэтому на официальной эмблеме языка изображена чашка с горячим кофе. Существует и другая версия происхождения названия языка, связанная с аллюзией на кофе-машину как пример бытового устройства, для программирования которого изначально язык создавался. В соответствии с этимологией в русскоязычной литературе с конца двадцатого и до первых лет двадцать первого века название языка нередко переводилось как Ява, а не транскрибировалось.

Начиная с середины 1990-х годов язык стал очень широко использоваться для разработки клиентских приложений и серверного ПО. Тогда же определённое распространение получила технология Java-апплетов — графических Java-приложений, которые встраивались в веб-страницы. С развитием возможностей динамических веб-страниц в 2000-е годы технология стала применяться редко.

В веб-разработке применяется Spring Framework, для документирования используется утилита Javadoc.

Python

Python (в русском языке встречаются названия пито́н или па́йтон) — высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью.

Язык ориентирован на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ. Язык полностью объектно-ориентированный. Необычной особенностью этого языка является отделение блоков кода пробельными отступами вместо точки с запятой. Синтаксис ядра языка минималистичен, за счёт чего редко возникает необходимость обращаться к документации, сам же язык известен как интерпретируемый и используется для написания скриптов. Недостатками языка являются более низкая скорость работы и более высокое потребление памяти написанных на нём программ по сравнению с аналогичным кодом, написанным на компилируемых языках, таких как Си или C++.

Python является языком, поддерживающим различные парадигмы:

* программирования: императивное, метапрограммирование
* процедурное
* структурное
* объектно-ориентированное программирование
* функциональное программирование.

Задачи обобщённого программирования решаются за счёт динамической типизации. Аспектно-ориентированное программирование частично поддерживается с помощью декораторов, более полноценная поддержка обеспечивается благодаря дополнительным фреймворкам. Такие методики как контрактное и логическое программирование можно реализовать с помощью библиотек или расширений.

Основные архитектурные черты языка — динамическая типизация, полная интроспекция, механизм обработки исключений, автоматическое управление памятью, поддержка многопоточных вычислений с глобальной блокировкой интерпретатора (GIL), высокоуровневые структуры данных. Поддерживается разбиение программ на модули, которые, в свою очередь, могут объединяться в пакеты.

Эталонной реализацией Python является интерпретатор CPython, который поддерживает большинство активно используемых платформ. Он является стандартом де-факто языка и распространяется под свободной лицензией Python Software Foundation License, которая позволяет использовать его без ограничений в любых приложениях, включая проприетарные. CPython компилирует исходные тексты в высокоуровневый байт-код, который исполняется в стековой виртуальной машине. К другим трём основным реализациям языка относятся Jython (для JVM), IronPython (для CLR/.NET) и PyPy. PyPy был написан на подмножестве языка Python (RPython) и разрабатывался в качестве альтернативы CPython, преследуя цель повышения скорости исполнения программ, в том числе за счёт использования JIT-компиляции. Поддержка версии Python 2 закончилась в 2020 году. На текущий момент активно развивается версия языка Python 3.

Стандартная библиотека языка включает большой набор полезных переносимых функций от функционала для работы с текстом и заканчивая средствами для написания сетевых приложений. Дополнительные возможности, такие как математическое моделирование, работа с оборудованием, написание веб-приложений или разработка игр, могут реализовываться с помощью огромного количества сторонних библиотек, а также с помощью интеграции библиотек, написанных на Си или C++, при этом и сам интерпретатор Python может интегрироваться в проекты, написанные на этих языках. Существует и специализированный репозиторий программного обеспечения, написанного на Python, — PyPI. Данный репозиторий предоставляет средства для простой установки пакетов в операционную систему и стал стандартом для Python. По состоянию на 2019 год в нём содержалось более 175 тысяч пакетов.

Python давно стал одним из самых популярных языков, он используется в анализе данных, машинном обучении, DevOps и веб-разработке, а также в других сферах, включая разработку игр. Благодаря читабельности, простому синтаксису и отсутствию необходимости в компиляции этот язык очень хорошо подходит для обучения программированию. Он позволяет концентрироваться на изучении алгоритмов, концептов и парадигм. Отладка и экспериментирование в сильно облегчаются тем фактом, что язык является интерпретируемым. Применяется язык многими крупными компаниями, такими как Google или Facebook. По состоянию на апрель 2021 года Python занимает третье место в рейтинге TIOBE популярности языков программирования с показателем 11,03 %. «Языком года» по версии TIOBE Python объявлялся в 2007, 2010, 2018 и 2020 году.

В итоге сравнительного анализа был выбран язык С++, как наиболее удовлетворяющий поставленным задачам для разработки ПМ НКСС. Выбор этого языка программирования продиктован требованиями и дальнейшем использованием модуля работниками ООО «С-Терра».

* 1. Выбор среды программирования.

Современная IDE для разработки на C++ должна поддерживать последнюю версию стандарта. В среде должна быть реализована поддержка отладчика и фреймворков для тестирования.

От IDE требуется настройка сборки под разные платформы, возможность работы с системой контроля версий Git. Разработка ПМ происходит на удаленной виртуальной машине, следовательно необходима возможность подключения по SSH.

Для выбранного языка был проведен анализ и выбор интегрированной среды разработки. Результаты сравнения представлены в таблице 2. Оценка проводилась по ключевым для разработки характеристикам, определяющим удобство написания ПО и скорость разработки.

Таблица 2

Сравнение сред разработки

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Eclipse | VS Code | Visual Studio | Code::Blocks | WinSCP |
| Возможность работы с Git | Есть | Есть | Есть | Есть | Нет |
| Удобство/опыт использования | Нет | Есть | Есть |  | Нет |
| Возможность подключения по SSH | Нет | Есть | Нет | Нет | Есть |
| Поддержка фреймворков для тестирования | Есть | Есть | Есть | Есть | Есть |
| Способ распространения | Бесплатная | Бесплатная | Бесплатная с ограничениями | Бесплатная | Бесплатная |

Ниже приведена подробная информация по каждой из представленных сред разработки

Eclipse

Eclipse — свободная интегрированная среда для разработки модульных кроссплатформенных приложений. Развивается и поддерживается Eclipse Foundation.

Наиболее известные приложения на основе Eclipse Platform — различные «Eclipse IDE» для разработки ПО на множестве языков. Наиболее популярный «Java IDE», который поддерживался изначально, не полагается на какие-либо закрытые расширения, использует стандартный открытый API для доступа к Eclipse Platform).

Eclipse – это в первую очередь платформа для разработки расширений, благодаря чему он и завоевал популярность. Любой разработчик может расширить Eclipse своими модулями. Уже существуют Java Development Tools (JDT), C/C++ Development Tools (CDT), разрабатываемые инженерами QNX совместно с IBM, и средства для языков Ada (GNATbench, Hibachi), COBOL, FORTRAN, PHP, X10 (X10DT) и пр. от различных разработчиков. Различные расширения дополняет среду Eclipse диспетчерами для работы с базами данных, серверами приложений и др.

Eclipse JDT — это наиболее известный модуль, нацеленный на групповую разработку. Среда разработки интегрирована с системами управления версиями — CVS, GIT в основной поставке, для других систем (например, Subversion, MS SourceSafe) существуют плагины. Также предлагается поддержка связи между IDE и системой управления задачами (ошибками). В основной поставке включена поддержка трекера ошибок Bugzilla. Кроме того, имеется множество различных расширений для поддержки других трекеров (Trac, Jira и др.). В силу бесплатности и высокого качества, Eclipse во многих организациях является корпоративным стандартом для разработки приложений.

Eclipse написана на Java, поэтому является продуктом, не зависящим от платформы, за исключением библиотеки SWT, которая разрабатывается для всех распространённых платформ. Библиотека SWT используется вместо стандартной для Java библиотеки Swing. Она целиком и полностью опирается на нижележащую платформу, что обеспечивает быстроту и натуральный внешний вид пользовательского интерфейса, но иногда она вызывает на разных платформах проблемы совместимости и устойчивости приложений.

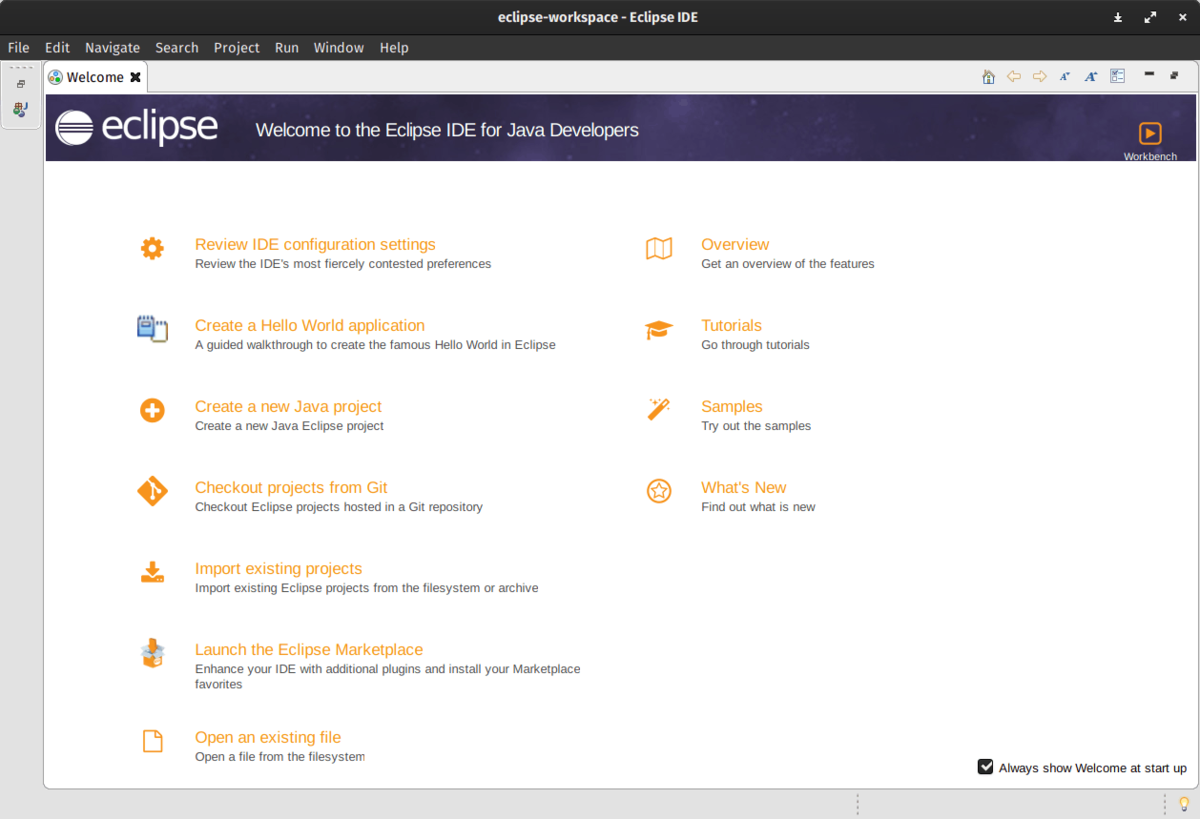


Рисунок 6 внешний вид Eclipse

Visual Studio Code

Visual Studio Code — это редактор исходного кода, разработанный компанией Microsoft для Windows, Linux и macOS. Позиционируется данный продукт как «лёгкий» редактор кода для кроссплатформенной разработки приложений. Включает в себя отладчик, инструменты для работы с Git, подсветку синтаксиса и средства для рефакторинга. Имеет довольно широкие возможности для кастомизации, такие как: пользовательские темы, сочетания клавиш и файлы конфигурации. Распространяется бесплатно, разрабатывается как ПО с открытым исходным кодом, но готовые сборки распространяются под проприетарной лицензией.

Многие возможности Visual Studio Code недоступны через графический интерфейс, зачастую они используются через палитру команд или JSON-файлы (например, пользовательские настройки). Палитра команд представляет собой подобие командной строки, которая вызывается сочетанием клавиш.

Visual Studio также позволяет заменять кодовую страницу при сохранении документа, символы перевода строки и язык программирования текущего документа.

С 2018 года появилось расширение Python для Visual Studio Code с открытым исходным кодом. Оно предоставляет разработчикам широкие возможности для редактирования, отладки и тестирования кода.

На март 2019 года посредством встроенного в продукт пользовательского интерфейса можно загрузить и установить несколько тысяч расширений только в категории «programming languages» (языки программирования).

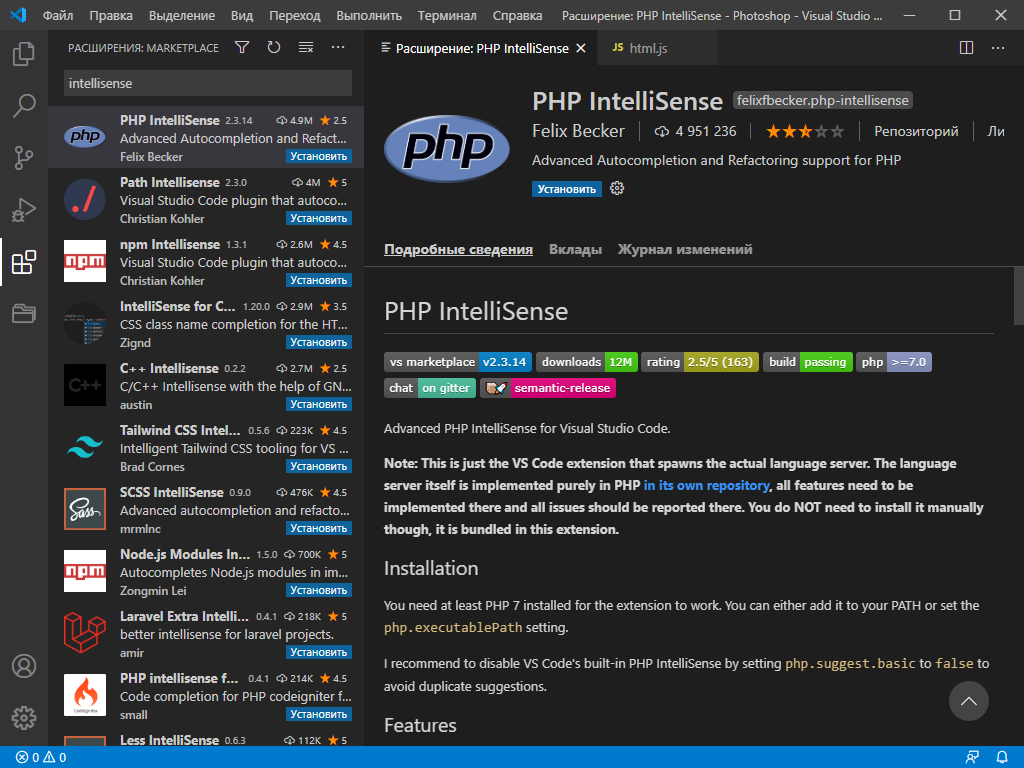


Рисунок 7Внешний вид Visual Studio Code

Visual Studio

Visual Studio — линейка продуктов от компании Microsoft. Они включают интегрированную среду разработки ПО и ряд других инструментальных средств. Данные продукты позволяют не только разрабатывать консольные приложения, но и игры или приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight.

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик вполне может работать в качестве отладчика уровня исходного кода, либо в качестве отладчика машинного уровня. Остальные встраиваемые инструменты могут включать в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет как создавать, так и подключать сторонние дополнения (плагины) для расширения функциональности среды разработки практически на каждом уровне. включая добавление поддержки систем контроля версий исходного кода (как, например, Subversion и Visual SourceSafe), добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения (например, клиент Team Explorer для работы с Team Foundation Server).

Некоторые из наиболее популярных дополнений:

* dbForge Fusion for Oracle
* dbForge Fusion for MySQL
* dbForge Fusion for SQL Server
* Review Assistant — плагин просмотра и редактирования кода
* AnkhSVN — свободная реализация клиента Subversion в Visual Studio (в настоящее время поддерживаются версии с 2005 по 2013).

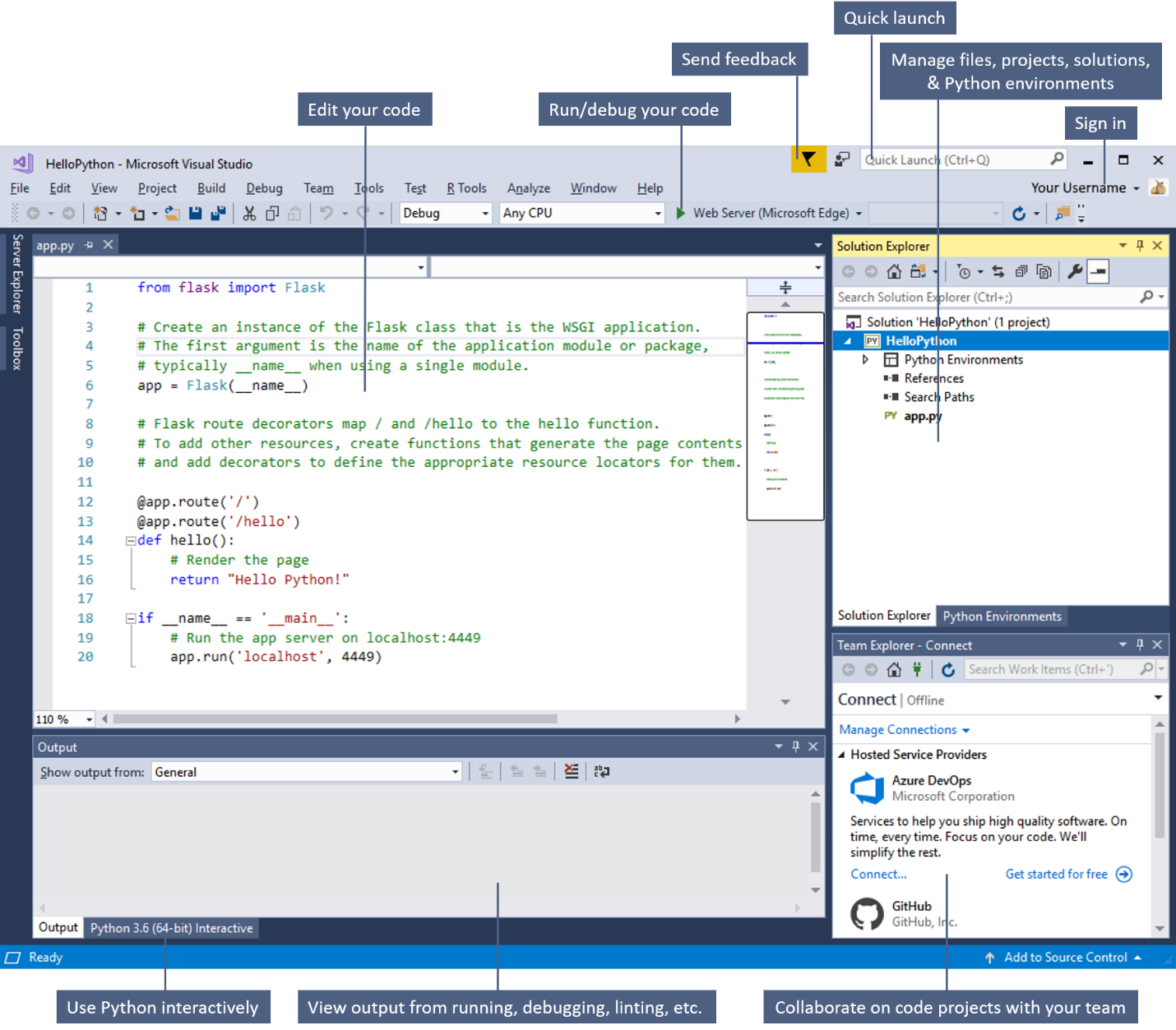


Рисунок 8 внешний вид MS Visual Studio

Code :: Blocks

Code :: Blocks - кросс-платформенная IDE c открытым исходным кодом, которая поддерживает несколько компиляторов , включая GCC, Clang и Visual C++ . Он разработан на C++ с использованием wxWidgets в качестве инструментария графического интерфейса. При использовании архитектуры плагина его возможности и функции определяются предоставленными плагинами. В настоящее время Code :: Blocks ориентирован на C , C ++ и Fortran . Он имеет настраиваемую систему сборки и дополнительную поддержку Make .

Code :: Blocks разрабатывается для Windows и Linux и был перенесен на FreeBSD , OpenBSD и Solaris. Последний двоичный файл для версии macOS - 13.12, выпущенный 12 декабря 2013 г. (совместим с Mac OS X 10.6 и более поздними версиями), но могут быть скомпилированы более свежие версии, и MacPorts поставляет версию 17.12.

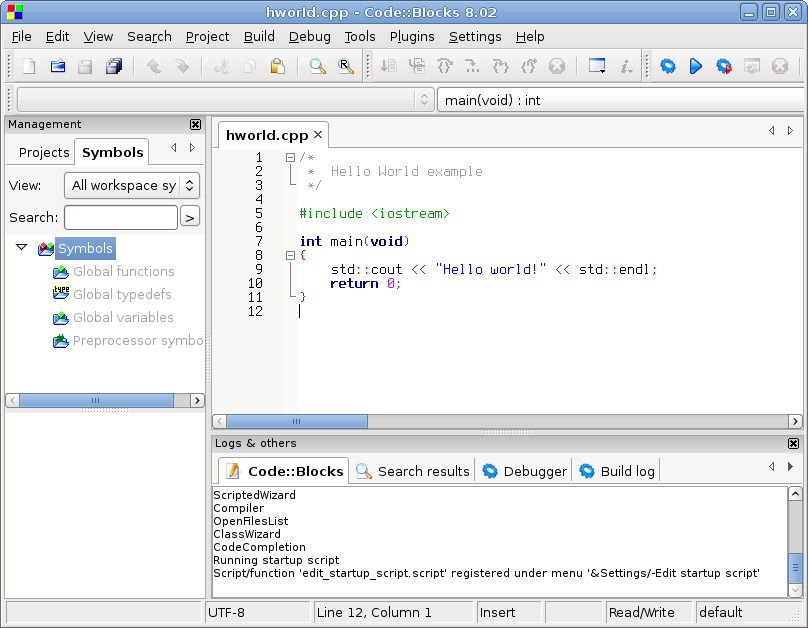


Рисунок 9внешний вид Code::Blocks

WinSCP

WinSCP — это свободно распространяемый графический клиент протоколов SFTP и SCP, работающий на Windows. Распространяется он по лицензии GNU GPL. Обеспечивает защищённое копирование различных файлов между компьютером и серверами, которые поддерживают эти протоколы.

Основные возможности:

* Графический интерфейс (GUI) в стиле Norton Commander и как в проводнике Windows Explorer (на выбор).
* Все самые основные файловые операции, такие как копирование, удаление и т. д.
* Автоматизация с использованием скриптов или интерфейса командной строки.
* Интеграция с Pageant (PuTTY Agent) с возможностью авторизации по открытым ключам.
* Интеграция с Windows (поддержка Drag&Drop, ярлыков, поддержка схем URL).
* Работа с ключами и версиями протокола SSH.
* Встроенный текстовый редактор и возможность выбора другого текстового редактора
* Поддержка различных типов авторизации, например: по паролю, аутентификации с открытым ключом, Kerberos.
* Возможность сохранить настройки соединений.
* Синхронизация папок по нескольким автоматическим и полуавтоматическим алгоритмам.
* Локализации интерфейса для нескольких десятков языков, в том числе русского.
* Возможность работы с использованием файла конфигурации вместо хранения настроек в реестре, что бывает очень удобно при запуске с переносных носителей.
* Поддержка протоколов SFTP и SCP поверх SSH-1 и SSH-2, а также FTP.
* Плагин для поддержки протокола SFTP в программе FAR Manager.

Другими словами, WinSCP не совсем среда разработки в чистом виде, скорее это просто клиент для более удобного подключения к серверу. Он позволяет менять файлы на сервере без использования командной строки, в том числе утилиты Vim. Поскольку для разработки требуется подключение по протоколу SCP данный вариант имеет один из высших приоритетов при выборе среды разработки.

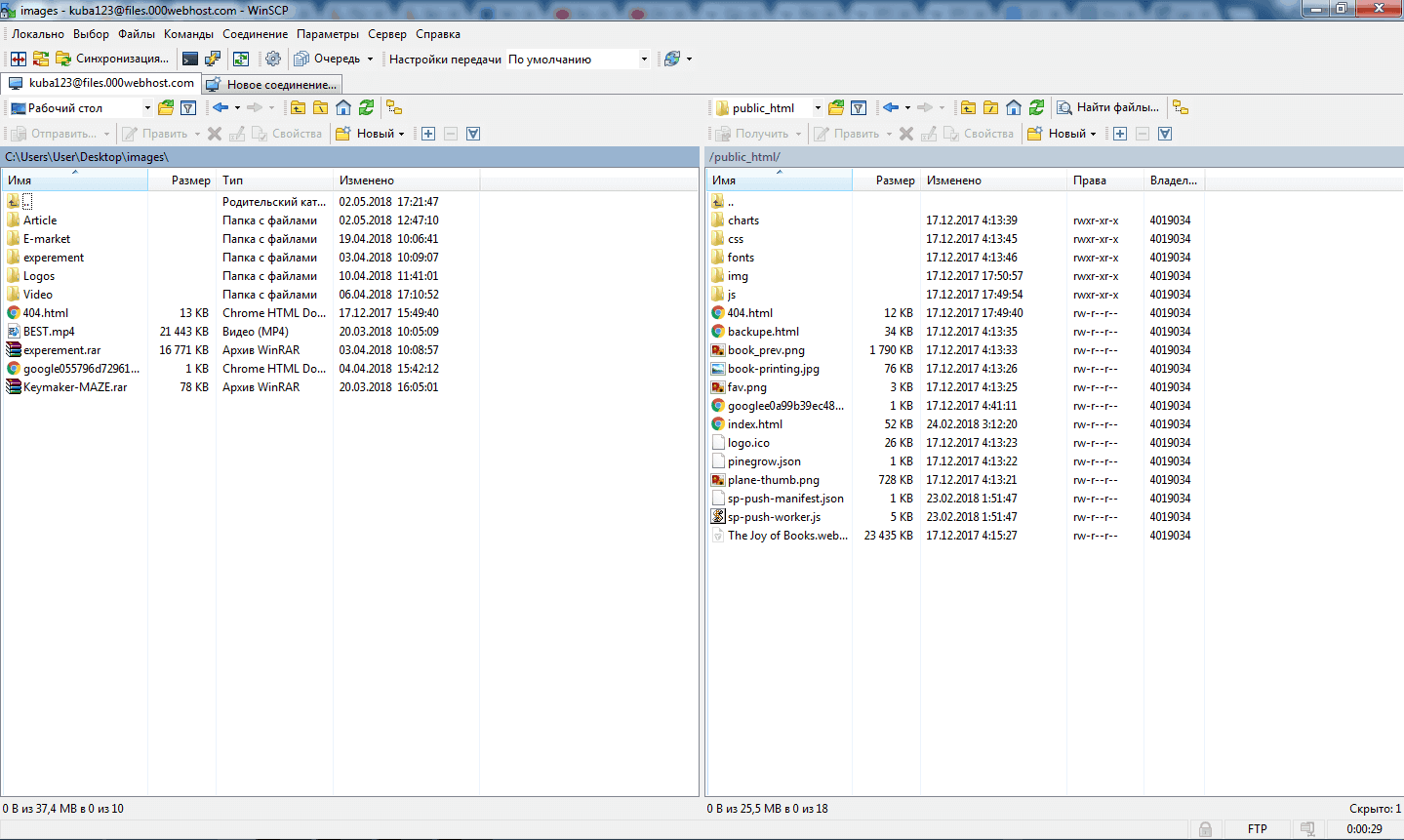


Рисунок 10 внешний вид WinSCP

# 2.3 Схема алгоритма

В начале работы ПМ НКСС получает на вход Cisco-like команду и ищет ее среди имеющихся в базе команд. В том случае, если введенная команда отсутствует, на консоль выводится сообщение об ошибке. Если же, такая команда есть проверяется, правильно ли введены данные для выполнения этой команды. В случае ошибки ПМ НКСС завершает свою работу и выводит сообщение об ошибке. Если же ошибок нет, то выполняются действия в зависимости от введенной команды. Команды могут быть на добавление нового NTP-сервера, на удаление одного из указанных серверов. В этом случае выполняются соответствующие настройки конфигурации. Также сетевой администратор может ввести команды на отображение текущего статуса. В этом случае в консоль выводятся запрошенные данные. Для наглядности, ниже представлена блок-схема алгоритма.

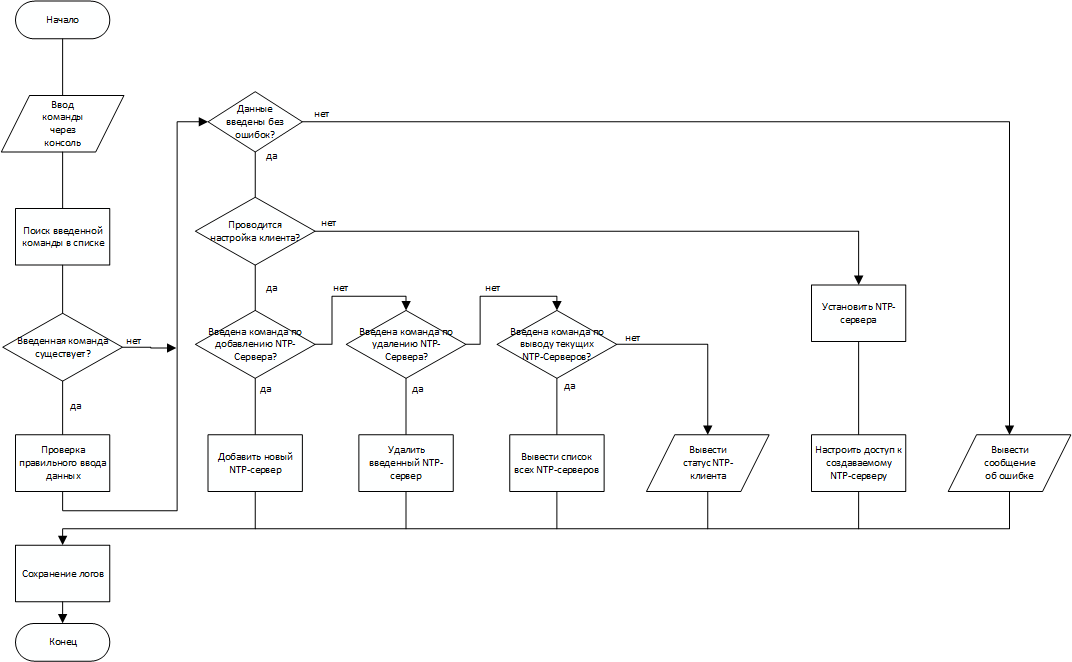


Рисунок 11 схема алгоритма

# 2.4 Организация связи с другими модулями.

Разрабатываемый ПМ НКСС является частью ПАК «С-Терра Шлюз» предназначен для упрощения настройки конфигурации NTP сервисов. Для своей работы он использует список команд, которые поддерживает ПАК, список модулей, реализующих эти команды, получает данные от модуля ввода данных, выводит данные с помощью модуля вывода и в зависимости от введенной команды и введенных данных, ПМ НКСС выставляет нужные настройки конфигурации.

Список команд, которые выполняет ПМ:

* Добавить новый NTP-сервер
* Удалить ранее введенный NTP-сервер
* Вывести в консоль список NTP-серверов
* Вывести на консоль статус NTP настроек



Рисунок 12схема данных

# Выводы по разделу

В конструкторском разделе был определен язык программирования для разработки, выбрана среда программирования.

Была спроектирована архитектура программного модуля, сформирован алгоритм работы.

# Технологический раздел

Тестирование программного обеспечения — это процесс исследования или испытания программного продукта. Цель тестирования – проверка соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определённым образом

Первые программные системы как правило разрабатывались в рамках программ научных исследований или программ для нужд министерств обороны. Тестирование таких продуктов было строго формализованным. Записывались все тестовые процедуры, тестовые данные, полученные результаты. Тестирование выделялось в отдельный процесс, который начинался после завершения кодирования, но при этом обычно выполнялось все тем же персоналом.

В 1960-х очень много внимания уделялось так называемому «исчерпывающему» тестированию. Оно должно было проводиться с использованием всех путей в коде или всех возможных входных данных. Было отмечено, что в таких условиях полное тестирование ПО невозможно, потому что, во-первых, количество возможных входных данных очень-очень велико, во-вторых, существует огромное множество путей, в-третьих, крайне сложно найти проблемы в архитектуре и спецификациях. По всем вышеперечисленным причинам «исчерпывающее» тестирование было отклонено и признано теоретически невозможным.

В начале 1970-х годов тестирование программного обеспечения обозначалось как «процесс, направленный на демонстрацию корректности продукта» или как «деятельность по подтверждению правильности работы программного обеспечения». В зарождавшейся программной инженерии верификация ПО значилась как «доказательство правильности». Хотя данная концепция была теоретически перспективной, на практике она требовала довольно много времени и была недостаточно всеобъемлющей. Было решено, что доказательство правильности — не особо эффективный метод тестирования ПО. Однако, в некоторых случаях демонстрация правильной работы используется и в наши дни, взять, например, приёмо-сдаточные испытания. Во второй половине 1970-х годов тестирование обычно представлялось как выполнение программы с намерением найти ошибки, а не доказать, что она работает. Успешный тест — это тест, который обнаруживает ранее неизвестные проблемы. Данный подход прямо противоположен предыдущему. Указанные два определения представляют собой «парадокс тестирования», в основе которого лежат два противоположных утверждения: с одной стороны, тестирование позволяет убедиться, что продукт работает хорошо, а с другой — выявляет ошибки в программах, показывая, что продукт не работает. Вторая цель тестирования является более продуктивной с точки зрения улучшения качества, так как не позволяет игнорировать недостатки программного обеспечения.

В 1980-е годы тестирование расширилось таким понятием, как предупреждение дефектов. Проектирование тестов — это наиболее эффективный из известных методов предупреждения ошибок. В это же время стали часто высказываться мысли о том, что необходима методология тестирования, в частности, что тестирование должно включать в себя проверки на всем протяжении цикла разработки, при этом, процесс должен быть управляемым. В ходе тестирования необходимо проверить не только собранную программу, но и требования, код, архитектуру, сами тесты. «Традиционное» тестирование, которое существовало до начала 1980-х годов, относилось только к скомпилированной, готовой системе (сейчас это в основном называется системное тестирование), но в дальнейшем все тестировщики стали вовлекаться во все аспекты жизненного цикла разработки. Это позволяло гораздо раньше находить проблемы в требованиях и архитектуре и благодаря этому сокращать сроки и бюджет разработки. В середине 1980-х годов появились самые первые инструменты для автоматизированного тестирования. Предполагалось, что компьютер сможет выполнить больше тестов, чем человек, и сделает это более надёжно. Поначалу эти инструменты были крайне простыми и не имели возможности написания сценариев на скриптовых языках.

В начале 1990-х годов в понятие «тестирование» стали включать планирование, проектирование, создание, поддержку и выполнение тестов и тестовых окружений, и это означало переход от тестирования к обеспечению качества, охватывающего весь цикл разработки программного обеспечения. В это время начинают появляться различные программные инструменты для поддержки процесса тестирования: более продвинутые среды для автоматизации с возможностью создания скриптов и генерации отчетов, системы управления тестами, ПО для проведения нагрузочного тестирования. В середине 1990-х годов с развитием Интернета и разработкой большого количества веб-приложений особую популярность стало получать «гибкое тестирование» (по аналогии с гибкими методологиями программирования).

Существуют различные классификации тестирования:

* По объекту тестирования (функциональное, производительности, безопасности, локализации и т. д.)
* По знанию внутреннего строения системы (черного/белого/серого ящика)
* По степени автоматизации (ручное, автоматическое, полуавтоматическое)
* По времени проведения тестирования (альфа, бета)

И множество других классификаций, для наглядности представленных ниже.

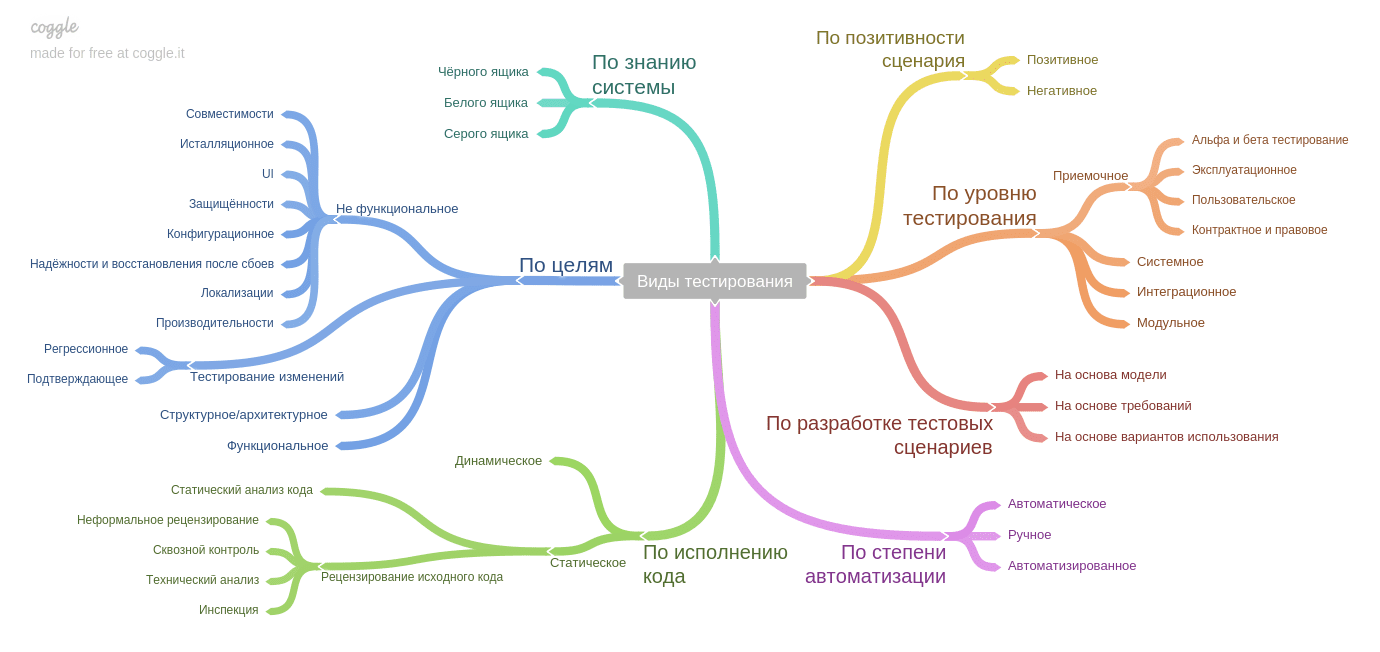


Рисунок 13виды тестирования

Разберем некоторые из методов тестирования.

# 3.1 Методы тестирования

Классификация по объекту тестирования.

Функциональное тестирование – это тестирование ПО в целях проверки реализуемости функциональных требований, то есть способности ПО в определенных условиях решать задачи, необходимые пользователям. Функциональные требования определяют, что именно делает ПО, какие задачи оно решает.

Тестирование производительности (англ. Performance Testing) в инженерии программного обеспечения — тестирование, которое проводится с целью определения, как быстро работает вычислительная система или её часть под определённой нагрузкой. Также может служить для проверки и подтверждения других атрибутов качества системы, таких как масштабируемость, надёжность и потребление ресурсов. Данный вид тестирования включает в себя нагрузочное тестирование, стресс-тестирование, тестирование стабильности, конфигурационное тестирование.

Юзабилити тестирование - исследование, выполняемое с целью определения, удобен ли некоторый искусственный объект (такой как веб-страница, пользовательский интерфейс или устройство) для его предполагаемого применения. Таким образом, проверка эргономичности измеряет эргономичность объекта или системы. Проверка эргономичности сосредоточена на определённом объекте или небольшом наборе объектов, в то время как исследования взаимодействия человек-компьютер в целом — формулируют универсальные принципы.

Тестирование безопасности — оценка уязвимости ПО к различным атакам. В ходе тестирования испытатель играет роль взломщика. Он может использовать любые средства от попыток узнать пароль с помощью внешних средств до просмотра несекретных данных в надежде найти ключ для входа в систему.

Также существует много тестов по объекту тестирования, но они узконаправлены, поэтому рассматриваться не будут.

Среди перечисленных выше методов тестирования самым важным является функциональное тестирование. Поэтому оно обязательно будет проводится для разрабатываемого ПМ НКСС. У модуля пользовательским интерфейсом является Cisco-подобная консоль, в которую пользователь вводит команды. Поэтому юзабилити-тестирования проводиться не будет. В рамках данной работы не поднимались вопросы безопасности приложения, так что этот параметр также не тестируется.

Классификация по уровню детализации

Модульное (unit) тестирование проверяет отдельные небольшие части приложения, которые могут функционировать отдельно от других частей. Как правило, тестируются отдельные функции или классы. Этот вид тестирования позволяет на начальном этапе разработки обнаружить многие ошибки.

Интеграционное тестирование уже направлено на проверку взаимодействия отдельных модулей друг с другом.

Системное тестирование проверяет приложение как единое целое, собранное из отдельных модулей.

Приемочное тестирование оценивает соответствие требований к программному продукту

Для ПМ НКСС будут использоваться все методы из данной классификации.

Классификация по знанию внутреннего строения системы

Тестирование чёрного ящика или поведенческое тестирование — стратегия (метод) тестирования функционального поведения объекта (программы, системы) с точки зрения внешнего мира, при котором не используется знание о внутреннем устройстве (коде) тестируемого объекта. Иначе говоря, тестированием чёрного ящика занимаются тестировщики, не имеющие доступ к исходному коду приложения. Под стратегией понимаются систематические методы отбора и создания тестов для тестового набора. Стратегия поведенческого теста исходит из технических требований и их спецификаций

Тестирование белого ящика (англ. white-box testing), также тестирование стеклянного ящика (англ. glass-box testing), структурное тестирование (англ. structural testing) — тестирование, которое учитывает внутренние механизмы системы или компонента (ISO/IEC/IEEE 24765). Обычно включает тестирование ветвей, маршрутов, операторов. При тестировании выбирают входы для выполнения разных частей кода и определяют ожидаемые результаты.

Тестирование серого ящика – это метод тестирования программного продукта или приложения с частичным знанием его внутреннего устройства. Для выполнения тестирования «серого ящика» нет необходимости в доступе тестировщика к исходному коду. Тесты пишутся на основе знания алгоритма, архитектуры, внутренних состояний или других высокоуровневых описаний поведения программы.

В данном случае разработчик также является тестировщиком, поэтому будет использоваться метод белого ящика.

Классификация по степени автоматизации

Ручное тестирование (manual testing) — часть процесса тестирования на этапе контроля качества в процессе разработки программного обеспечения. Оно производится тестировщиком без использования программных средств, для проверки программы или сайта путём моделирования действий пользователя. В роли тестировщиков могут выступать и обычные пользователи, сообщая разработчикам о найденных ошибках.

Автоматизированное тестирование программного обеспечения — часть процесса тестирования на этапе контроля качества в процессе разработки программного обеспечения. Оно использует программные средства для выполнения тестов и проверки результатов выполнения, что помогает сократить время тестирования и упростить его процесс.

Для данного ПМ НКСС будет использоваться как ручное, так и автоматическое тестирование.

# 3.2 Выбор средств для тестирования ПМ НКСС

Для выбора рассмотрим средства тестирования для НКСС

* функциональное тестирование;
* метод белого и черного ящика;
* модульное, интеграционное и системное тестирование;
* автоматизированное и ручное.

Средства и методы отладки программы

Существует несколько методов отладки, подразделяемые на статические и динамические, которые применяются в зависимости от требуемых условий и конкретной реализации программы.

Статические методы – это методы, для выполнения которых не требуется запуск и исполнение программы. Эти методы требуют значительных временных затрат от программиста при незначительных затратах вычислительных мощностей. К статическим методам отладки программ относят прокрутку программы вручную или с помощью программных анализаторов (например, компилятор), а также коллективную проверку кода программы и проверку кода программистом-технологом.

Экспериментально установлено, что в программах ручными методами удается обнаруживать от 30 до 70 % программных и алгоритмических ошибок из общего числа ошибок, выявленных при отладке. При этом одновременно осуществляется доработка программ с целью улучшения их структуры, логики обработки данных и для снижения сложности последующего автомати­зированного тестирования на РС.

В динамических методах используются в большей степени вычислительные мощности, и отладка происходит совместно с исполнением программы. Данный подвид методов отладки, как правило, привязан к конкретному транслятору (компилятору), либо к среде разработки. К динамическим методам отладки относят такие методы, как поиск ошибок с использованием сторонних системных средств, а также использование специальных отладчиков.

# 3.3 Прототипирование

Прототипирование – проверка функций (модулей, библиотек, и т.п.) в изоляции с помощью небольших примеров кода (прототипов). Прототип легче отлаживать, чем целевую систему. Если проблема воспроизводиться с помощью прототипа, отладка упрощается. Unit тестирование в этом смысле более эффективный метод отладки, поскольку unit test-ы выполняются автоматически и «накапливаются» для будущего реюза, а прототипы редко становятся частью системы.

Качества, которыми должен обладать эффективный прототип:

* Этап создания прототипа не должен быть затяжным.
* Эффективные прототипы являются одноразовыми. Они предназначены для того, чтобы донести идею до заинтересованного лица. После того как идея была донесена, прототип может быть отвергнут.
* Эффективные прототипы являются сфокусированными, это означает что следует обращать внимание на сложные части при создании прототипов.
* Необходимо обращать внимание на элементы взаимодействия, которые принесут пользу вашему продукту.

# 3.4 Отладка по точкам останова

В программировании то́чка остано́ва (англ. breakpoint) — это преднамеренное прерывание выполнения программы, при котором выполняется вызов отладчика (одновременно с этим программа сама может использовать точки останова для своих нужд). После перехода к отладчику программист может исследовать состояние программы (логи, состояние памяти, регистров процессора, стека и т. п.), с тем чтобы определить, правильно ли ведёт себя программа. В отличие от полной остановки, с помощью останова, после работы в отладчике программа может быть завершена либо продолжена с того же места, где произошёл останов.

На практике точка останова определяется как одно или несколько условий, при которых происходит прерывание программы. Наиболее часто используется условие останова при переходе управления к указанной инструкции программы (instruction breakpoint). Другое условие останова — операция чтения, записи или изменения указанной ячейки или диапазона ячеек памяти (data breakpoint или watchpoint).

В процессе разработки и отладки ПМ НКСС активно использовался такой динамический метод отладки, как отладка с использованием точек останова. Для выполнения данного метода отладки был использован стандартный отладчик, встроенный в среду разработки Visual Studio Code.

Суть данного метода отладки заключается в размещении специальных точек останова в местах, в которых необходимо проконтролировать состояние программы. При запуске программы в режиме отладки с точками останова программа будет выполняться без изменений до тех пор, пока не дойдет до точки останова. При достижении точки штатное выполнение работы программы приостанавливается и управление передается на отладчик среды программирования. После этого становится возможным отследить состояния различных параметров программы на текущий момент, включая значения переменных и стек вызовов. На рисунке ниже изображено как выглядит отладка по точкам останова.

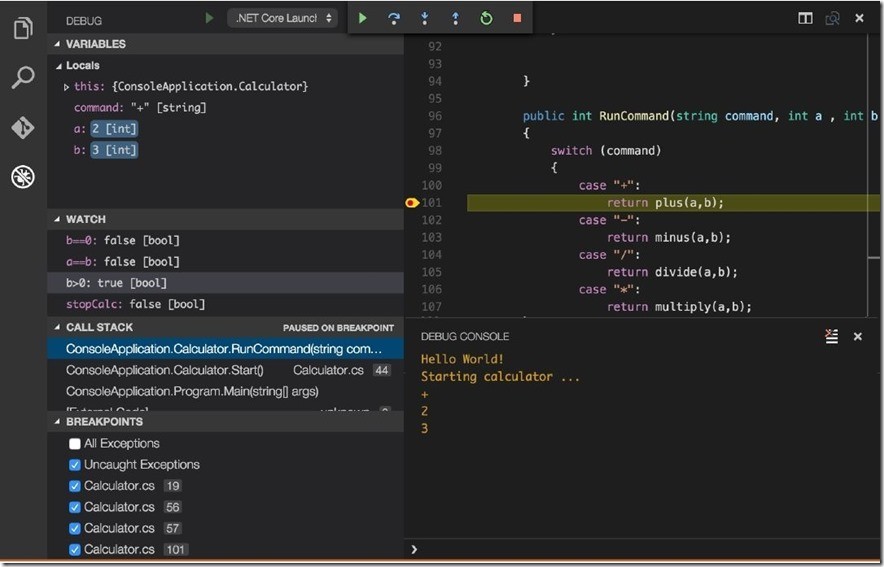


Рисунок 14 отладка в VS Code

# 3.5 Отладка с помощью трассировки

Трассировка – это метод отладки, при котором работа программы выполняется пошагово с шагом по строкам. Разработчик контролирует каждый шаг выполнения программы, стремясь полностью отследить её поведение.

При трассировке, изображенной на рисунке ниже, разработчик располагает следующими командами контроля исполнения программы:

шаг внутрь (англ. Step Into) – программа выполнит текущую строчку кода и курсор перейдет на следующую;

принудительный шаг внутрь (англ. Force Step Into) – программа выполнит текущую строчку кода и курсор перейдет на следующую игнорируя все возможные ограничения;

шаг с обходом (англ. Step Over) – практически аналогичен предыдущей команде, но если текущая строка подразумевает вход в отдельно описанную процедуру, то код, содержащийся в ней, выполнится без остановок;

выполнить до курсора(англ. Run to Cursor) – программа будет выполнять без остановки все строки, пока не дойдет до строки, на которой находится курсор.

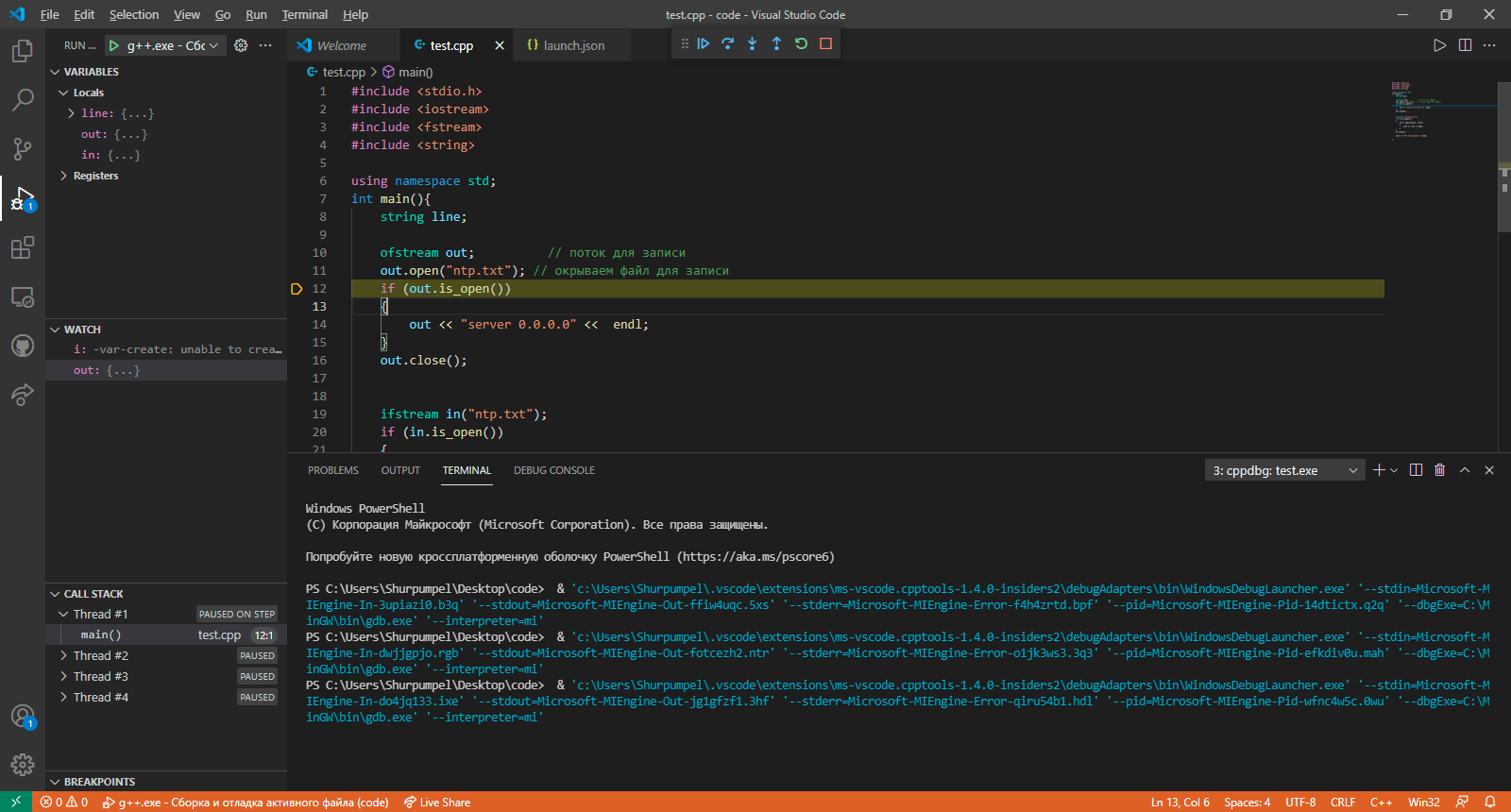


Рисунок 15трассировка в VS Code

# 3.6 Отладка ПМ НКСС

Ошибки, возникающие при разработке программного обеспечения, можно разделить на синтаксические и логические.

Синтаксические – это такие ошибки, которые не позволяют транслятору или интерпретатору однозначно интерпретировать написанный исходный код программы. Иначе говоря, это ошибки в записи конструкций языка программирования. Зачастую такие ошибки вызваны обычными опечатками.

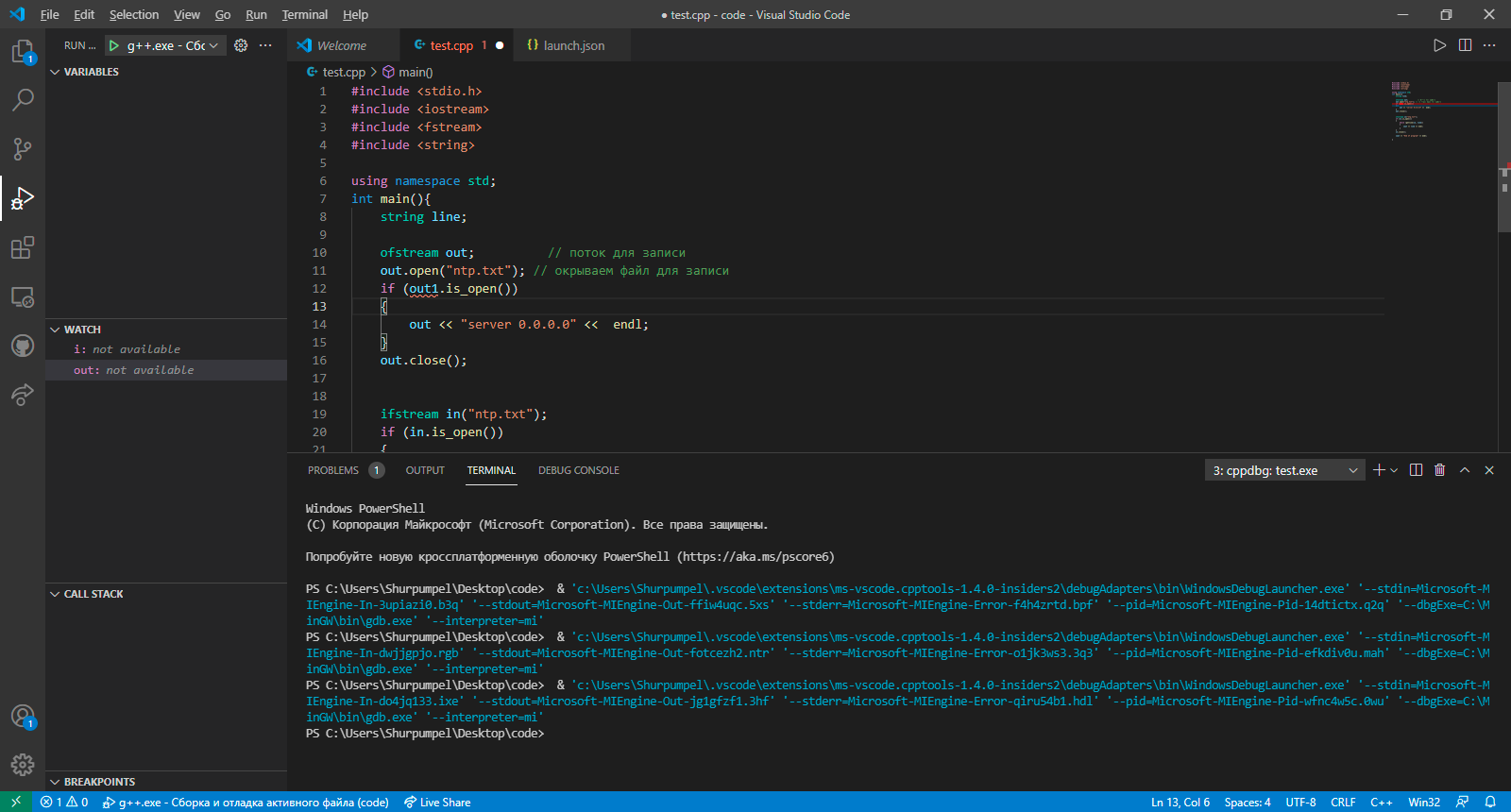
В случае с разработкой ПМ НКССС подобные проблемы выявляет встроенный анализатор среды разработки VS Code еще до трансляции кода и сборки проекта. Среда разработки указывает программисту файл, в котором обнаружена ошибка, ее расположение в коде этого файла, выделяет проблемное место и показывает подробную информацию во всплывающей подсказке при наведении курсора на это место. В качестве проблемных мест, говоря о средах разработки, могут быть следующие ошибки:

* ошибки подключения библиотек;
* ссылки на ненайденные файлы;
* использование необъявленных или неинициализированных переменных;
* ошибки построения циклов, условий и прочих стандартных структур;
* несоответствие типов данных при работе со строгой типизацией.

В случае если данный анализатор что-либо пропустит или если программист проигнорирует его предупреждения, интерпретатор завершит свою работу с ошибкой.

Пример работы синтаксического анализатора в VS Code показан на рисунке ниже.

Таким образом, выявление синтаксических ошибок является тривиальной задачей и не требует подробного рассмотрения.



Логические ошибки – это любые ошибки, которые приводят к неверным результатам работы программы, при этом код может выполняться без сбоев и не приводить к аварийной остановке. Наличие именно таких ошибок в коде выявляется на этапе тестирования программного обеспечения. Для локализации ошибок и нахождения причин их возникновения применяется анализ путей выполнения программы и текущих значений переменных. Данный анализ производится при помощи отладочного вывода и специальных инструментов – отладчиков, которые позволяют пошагово выполнять программу и отслеживать состояние переменных в памяти.

# 3.7 Модульное тестирование

Модульное тестирование, или юнит-тестирование (англ. unit testing) — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Специализированных утилит для создания Unit-тестов для языка С++ нет, поэтому было принято решение написать небольшой простенький ПМ, который будет проводить данное тестирование. Тесты отслеживают, как ведет себя ПМ НКСС при различных введенных данных. Проверяется поведение ПМ при вводе неправильных данных команды, при вводе правильных данных, при различных данных для команды.

Cписок разработанных тестов

#include <stdio.h>

#include <iostream>

#include <fstream>

#include <string>

#include <cstring>

#include <cassert>

#include "test.h"

using namespace std;

void TestShowRun();

void TestNTPServer();

void TestShowNTPStatus();

void TestShowNTPAssociations();

bool FindString(string ss);

int main()

{

    TestShowRun();

    TestNTPServer();

    TestShowNTPStatus();

    TestShowNTPAssociations();

    cout << "All tests OK" << endl;

    return 0;

}

void TestShowRun()

{

    system("configure terminal");

    system("ntp server 0.0.0.0");

    system("show run | include ntp");

    bool ff = strcpy(m\_LastResult, "ntp server 0.0.0.0");

    assert(ff == true);

    system("ntp server 0.0.0.0.0");

    system("show run | include ntp");

    bool ff = strcpy(m\_LastResult, "");

    assert(ff == true);

}

void TestNTPServer()

{

    system("configure terminal");

    system("ntp server 0.0.0.0");

    bool ff = FindString("server 0.0.0.0");

    assert(ff == true);

    system("no ntp server 0.0.0.0");

    ff = FindString("server 0.0.0.0");

    assert(ff == false);

    system("ntp server 0.0.0.0.0");

    ff = FindString("server 0.0.0.0");

    assert(ff == false);

    system("ntp server 0.0.0.0");

    system("no ntp server 0.0.0.0");

    ff = FindString("server 0.0.0.0");

    assert(ff == false);

}

void TestShowNTPAssociations()

{

    system("configure terminal");

    system("ntp server 0.0.0.0");

    bool ff = false;

    char\* a = strstr(m\_LastResult,"0.0.0.0");

    if(a != NULL)

    {

        ff = true;

    }

    assert(ff == true);

}

bool FindString(string ss)

{

    ifstream in("ntp.conf");

    string line;

    bool isThisStringInConfig = false;

    if (in.is\_open())

    {

        while (getline(in, line))

        {

            char str[line.length()+1];

            strcpy(str, line.c\_str());

            char str1[line.length()+1];

            strcpy(str1, line.c\_str());

            if (strcmp(str1, str)==0)

            {

                isThisStringInConfig = true;

            }

        }

    }

    in.close();

    return isThisStringInConfig;

}

В результате выполнения ПМ предназначенного все тесты были пройдены

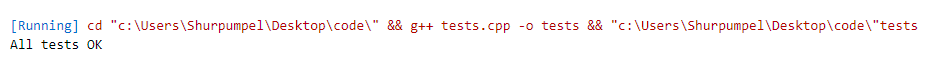


Рисунок 16 результаты Unit-тестирования

# Выводы по разделу

В данном разделе были приведены краткие описания методов тестирования. Из них выбраны наиболее подходящие для обеих частей ПМ НКСС. Описаны используемые для тестирования и отладки инструменты, процесс и результаты.

# Заключение

Данная работа была посвящена разработке программного модуля настройки конфигурации NTP сервиса. В ходе работы были выполнены следующие задачи:

* проведено исследование предметной области;
* проведен обзор существующих решений;
* выбор языка и среды программирования;
* разработана структура данных ПМ НКСС;
* разработан алгоритм работы ПМ НКСС;
* осуществлена программная реализация ПМ НКСС;
* проведены тестирование и отладка ПМ НКСС;
* разработано руководство оператора ПМ НКСС.

Цель разработки достигнута.