МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и технологий

**Отчет по лабораторной работе № 4**

по дисциплине: «Системное программирование»

на тему: «Автоматическая сборка многофайловых проектов»

Выполнила: ст. гр. 10702221

Баяманова А.К.

Проверил: Давыденко Н.В.

Минск 2023

**ЛАБОРАТОРНАЯ РАБОТА #4**

**Автоматическая сборка многофайловых проектов**

**Цель работы**

Научиться эффективно использовать специальные средства для автоматизации процесса компиляции, сборки и запуска многофайловых проектов.

**Требования**

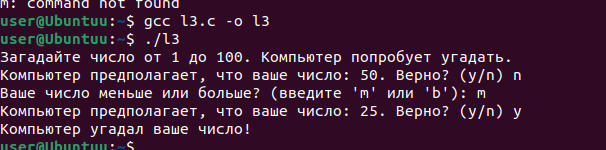
1) Для автоматизации сборки проекта необходимо использовать стандартную оболочку сценариев Linux bash (/bin/sh), а в качестве утилиты автосборщика – GNU make.

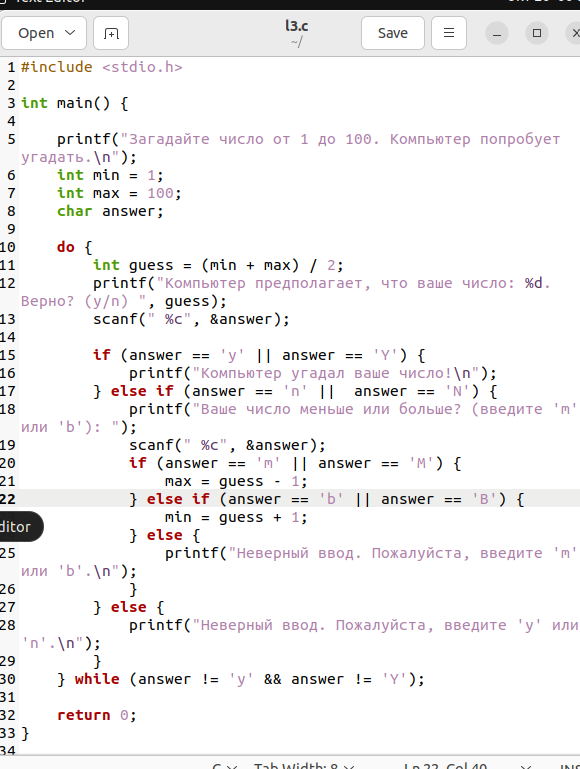
2) Для автосборки проекта с использованием GNU make необходимо описать два типа make-файлов: простой и параметризированный.

3) При выполнении задания запрещается использовать IDE. Рекомендуетсязадействовать любой текстовый редактор (к примеру, gedit) и набор компиляторов GNU Compiler Collection (GCC), в частности, компиляторы языков программирования С/C++ gcc/g++, а для отладки – gdb-отладчик.

**Основное задание**

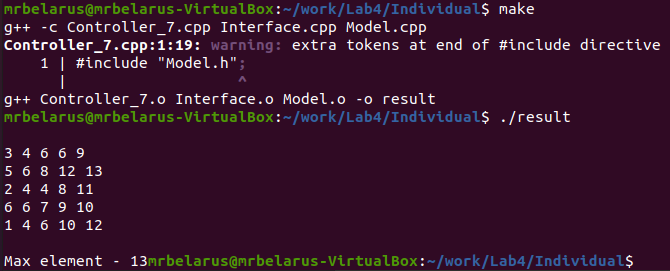
Необходимо переработать основное задание из предыдущей лабораторной работы таким образом, чтобы пользователь (игрок) загадывал число, а компьютер, используя оптимальный и эффективный алгоритм, его отгадывал.





**Индивидуальное задание**

Для программ, которые были разработаны в двух предыдущих лабораторных работах №2 и №3 необходимо добавить автоматизацию сборки многофайлового проекта с использованием сценарных оболочек и автосборщиков.



**Контрольные вопросы**

**Какие существуют способы и средства для автоматической сборки многофайловых проектов? Опишите категории утилит автосборок в Linux, а также их преимущества и недостатки?**

Make — мастодонт и заслуженный ветеран систем сборки, которого все никак не хотят отпустить на пенсию, а заставляют везти на себе все новые и новые проекты. Это очень низкоуровневая тулза со своим специфичном языком, где за пробел вместо таба вам сразу же грозит расстрел на месте. С помощью make можно сделать все, что угодно — билд любой сложности, но за это придется заплатить усилиями для написания скрипта, а также его поддержки в актуальном состоянии. Переносить логику билда из проекта в проект также будет накладно.

CMake — первая попытка уйти от низкоуровневых деталей make-а. Но движком здесь служит все тот же make для которого CMake генерирует огромные make-файлы на основе другого текстового файла с более выскоуровневым описанием билда.

Ant — своеобразный клон make для Java.

SCons — самодостаточная, кросплатформенная билд система, написанная на Python. SCons одинаково хорошо справляется как с Java так и с C++ билдами. Зависимости хидеров для инкрементальной сборки отрабатываются корректно (насколько я понял создается некая база данных с метаданными билда), а на Windows «без бубна» работает MSVC. Язык сценария сборки — Python..

Нет никаких абстрактных настроек для компилятора, поэтому если, например, возникнет необходимость сменить тулчейн, возможно, понадобиться искать места в билд скрипте для внесения изменений. Те же макросы придется прописывать с вложенными условиями — если это Виндовс то сделай так, если это GCC сделай так и т.д.

Нет поддержки удаленных артефакториев и высокоуровневой зависимости одного билда на другой.

Общая архитектура построена так, что так называемые user defined builders существуют практически изолированно и нет возможности заиспользовать уже существующую логику билда, чтобы дополнить ее своей через несложный плагин. Но в целом это достойный выбор для небольших проектов.

Gradle

Для JVM языков в Gradle очень удобная концепция работы с библиотеками, необходимыми для построения проекта (билд зависимостями):

В скрипте прописываются адреса репозиториев с артефактами: maven или ivy — например. Так же это может быть репозиторий любого другого типа/формата — лишь бы был плагин для него. Это может быть удаленный репозиторий, какой-нибудь Maven Central или ваш личный хостинг где-нибудь в сети или просто локальная репа на файловой системе.

Так же в специальном разделе скрипта указываются непосредственно зависимости для построения — список необходимых бинарных артефактов с указанием версий.

Перед началом построения Gradle пытается зарезолвить все зависимости и ищет артефакты с заданными версиями по всем репозиториям. Бинарники загружаются в кэш и автоматически добавляются в билд. Это очень удобно и я надеялся, что для C++, возможно, сделали нечто подобное.