Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Системное программирование

Студент: Лопатнюк П.В.

ФИТ 3 курс 1 группа

Преподаватель: Бернацкий П.В.

Минск 2024

**Лабораторная работа №1**

**Цель работы:** Получение практических навыков в работе с инструментом автоматической сборки проектов CMake и транслятором Clang. А также закрепление теоретических знаний о этапах разработки программ.

**Приложение Lab-01C:** Разработать приложение на языке C содержащее решение для задачи: «Дано вещественное число x. Не пользуясь никакими операциями, кроме умножения, получить x2, x5 и x17 за шесть операций». При чём код должен быть кроссплатформенным (т.е. компилироваться для Windows и Linux/macOS без внесения дополнительных изменений).

|  |
| --- |
| #include <stdio.h>  int main() {  double x;  printf("Type the number x: ");  scanf("%lf", &x);  double x2 = x \* x;  double x4 = x2 \* x2;  double x8 = x4 \* x4;  double x16 = x8 \* x8;  double x17 = x16 \* x;  double x5 = x4 \* x;  printf("x^2 = %.6f\n", x2);  printf("x^5 = %.6f\n", x5);  printf("x^17 = %.6f\n", x17);  return 0;  } |

**Приложение Lab-01CS:** Разработать приложение аналогичное первому, но на языке C#.

|  |
| --- |
| using System;  class Program  {  static void Main()  {  Console.Write("Enter number x: ");  double x = double.Parse(Console.ReadLine());  double x2 = x \* x;  double x4 = x2 \* x2;  double x8 = x4 \* x4;  double x16 = x8 \* x8;  double x17 = x16 \* x;  double x5 = x4 \* x;  Console.WriteLine($"x^2 = {x2}");  Console.WriteLine($"x^5 = {x5}");  Console.WriteLine($"x^17 = {x17}");  }  } |

Подготовить исполняемые файлы, полученные двумя разными способами, а именно:

с использованием транслятора Clang и компилятора С# напрямую;

с использованием CMake.

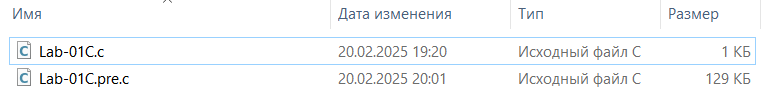
Данный пункт касается только приложения Lab-01C!!! При использовании Clang напрямую должны быть получены промежуточные файлы сборки, т.е. файлы, которые получаются на выходе каждого этапа сборки (после препроцессинга, после компиляции, после компоновки).

**Процесс компиляции приложения на языке C с Clang**

**1. Препроцессинг** (предварительная обработка)

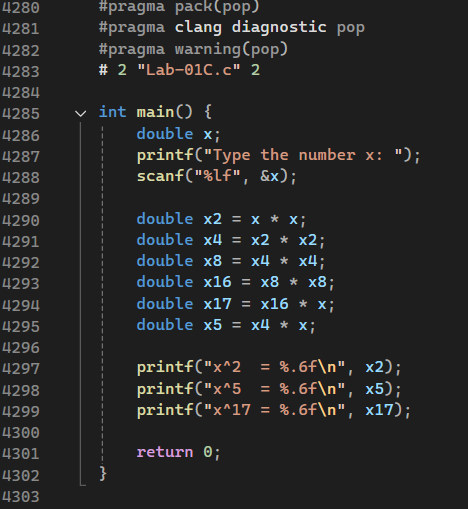
clang -E Lab-01C.c -o Lab-01C.pre.c





Обрабатывает директивы #include, #define, заменяет макросы. Удаляет комментарии. Не выполняет компиляцию.

**Результат**: Lab-01C.pre.c - код после обработки препроцессором.



**2. Анализ кода**

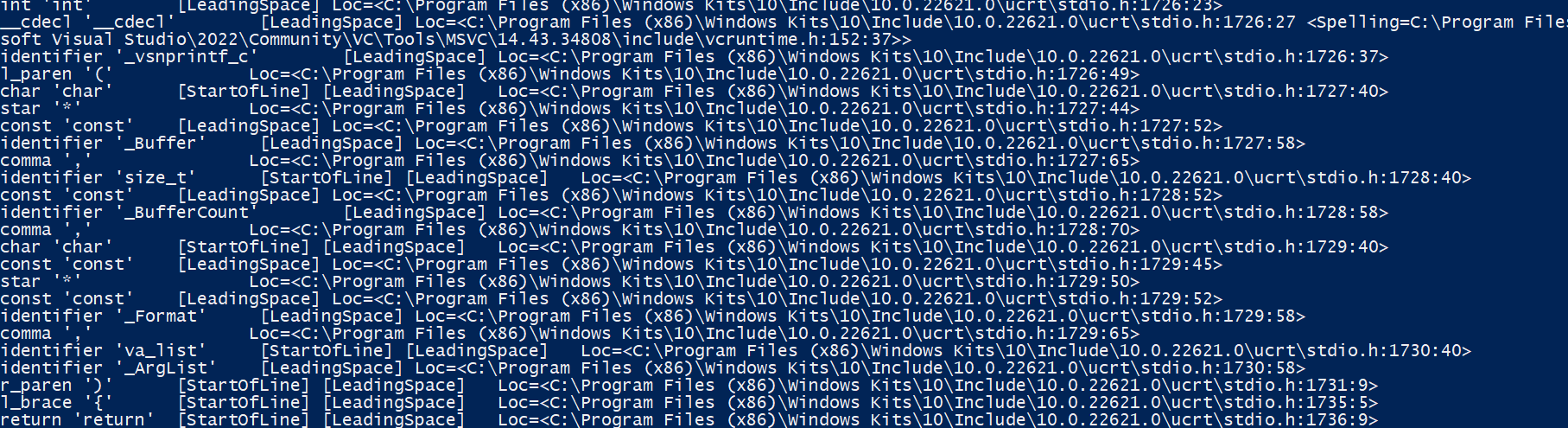
**2.1. Лексический анализ** (токены)

clang -fsyntax-only -Xclang -dump-tokens Lab-01C.c



Разбирает код на отдельные элементы (токены). Показывает, как Clang видит переменные, операторы, ключевые слова.

**Результат:** список токенов в консоли.



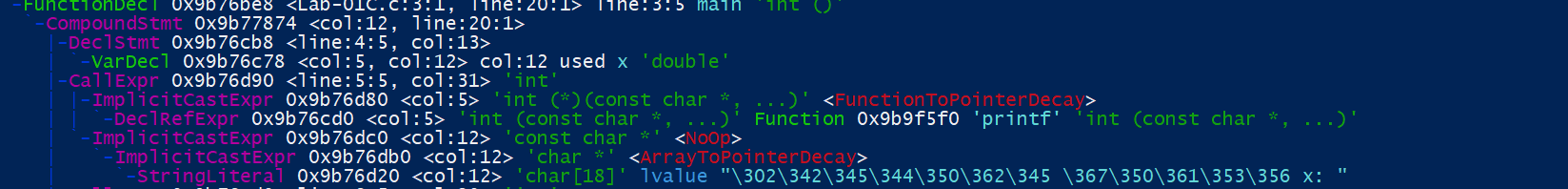
**2.2. Синтаксический анализ** (AST)

clang -fsyntax-only -Xclang -ast-dump Lab-01C.c



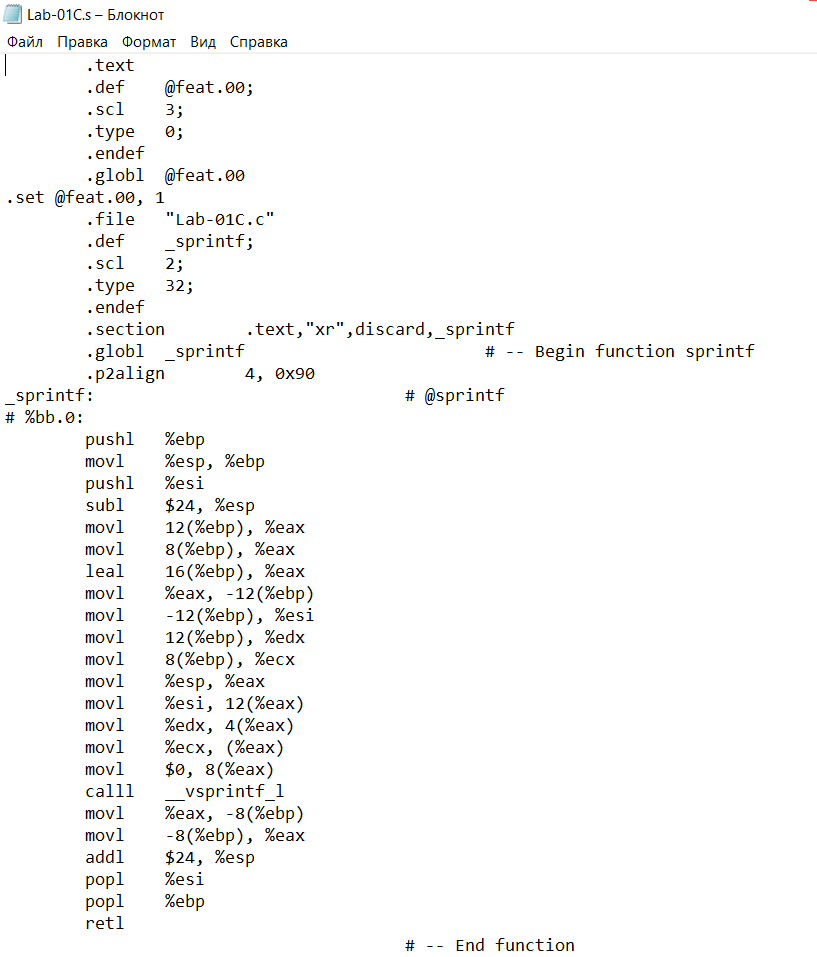
Строит абстрактное синтаксическое дерево (AST). Показывает, как Clang интерпретирует код до генерации машинных команд.

**Результат:** дерево AST в консоли.



**2.3 Ассемблерный код**



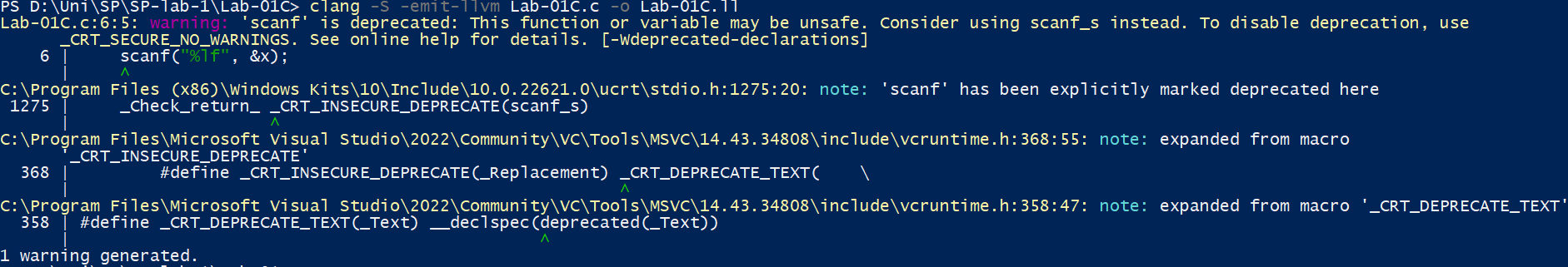


**3. Генерация промежуточного кода** (LLVM IR)

**Без оптимизации**

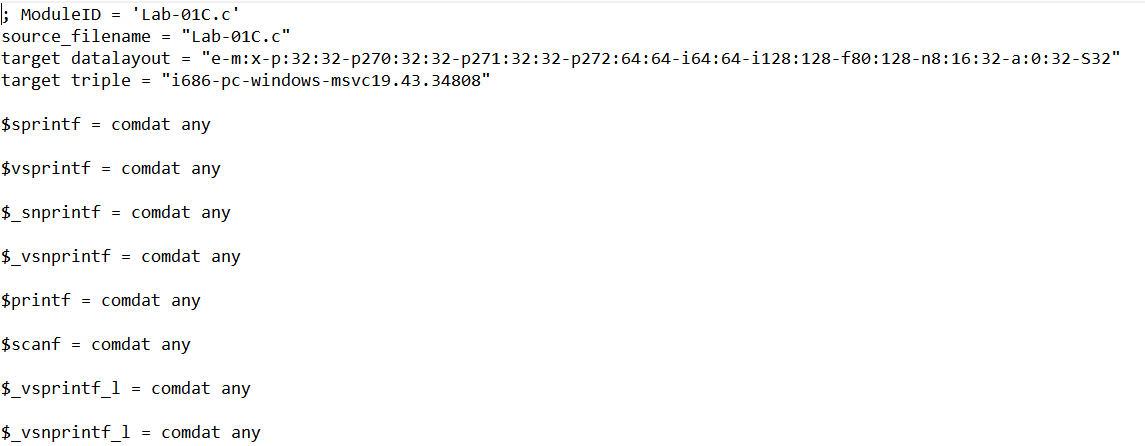
clang -S -emit-llvm Lab-01C.c -o Lab-01C.ll



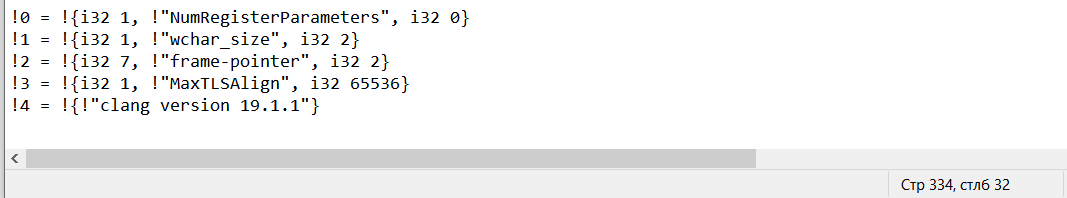


Генерирует LLVM IR (байткод) — промежуточное представление перед компиляцией в машинный код.

**Результат:** файл Lab-01C.ll с байткодом.



--



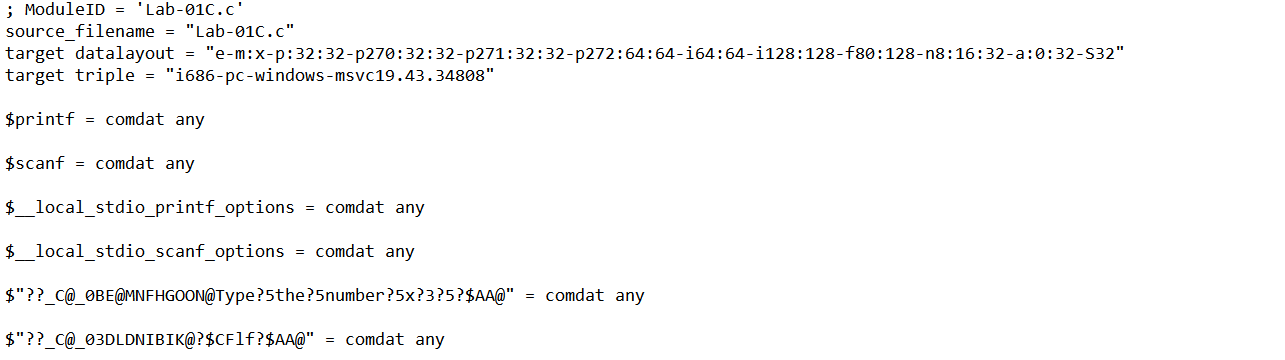
**С оптимизацией (-O3)**

clang -S -O3 -emit-llvm Lab-01C.c -o Lab-01C\_opt.ll

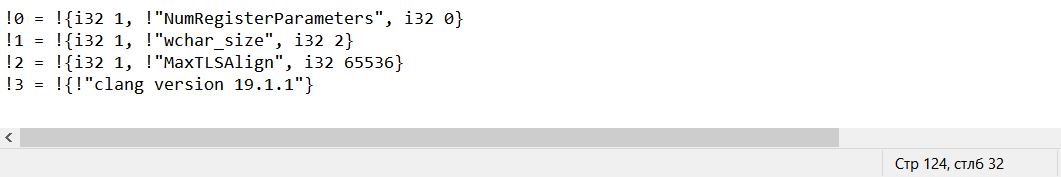


Выполняет агрессивные оптимизации. Упрощает выражения, удаляет лишние переменные.

**Результат:** Lab-01C\_opt.ll — оптимизированный LLVM IR.



--



LLVM IR (Lab-01C.ll) — это не C-код, а промежуточное представление.  
Его нельзя передавать напрямую компилятору, который ждёт C-код. Если мы хотим использовать main.ll вместо main.c, то нам нужно не clang -c, а llc или clang с опцией -c.

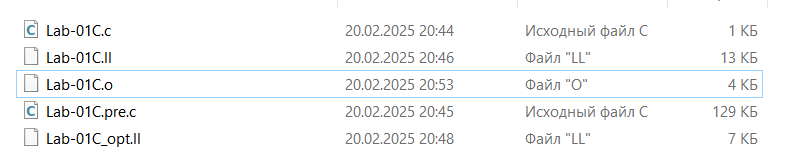
1. **Компиляция в объектный файл**

clang -c Lab-01C.c -o Lab-01C.o



Преобразует код в объектный файл (.o). Это ещё не исполняемый файл.

**Результат:** Lab-01C.o — машинный код без компоновки.



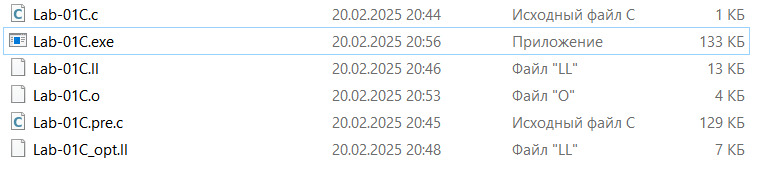
1. **Компоновка в исполняемый файл**

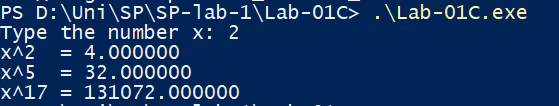
clang Lab-01C.o -o Lab-01C.exe



Соединяет объектный файл с библиотеками. Создаёт готовую программу.

**Результат:** Lab-01C.exe (Windows) или a.out (Linux/macOS).





**Процесс компиляции приложения на C (Linux)**

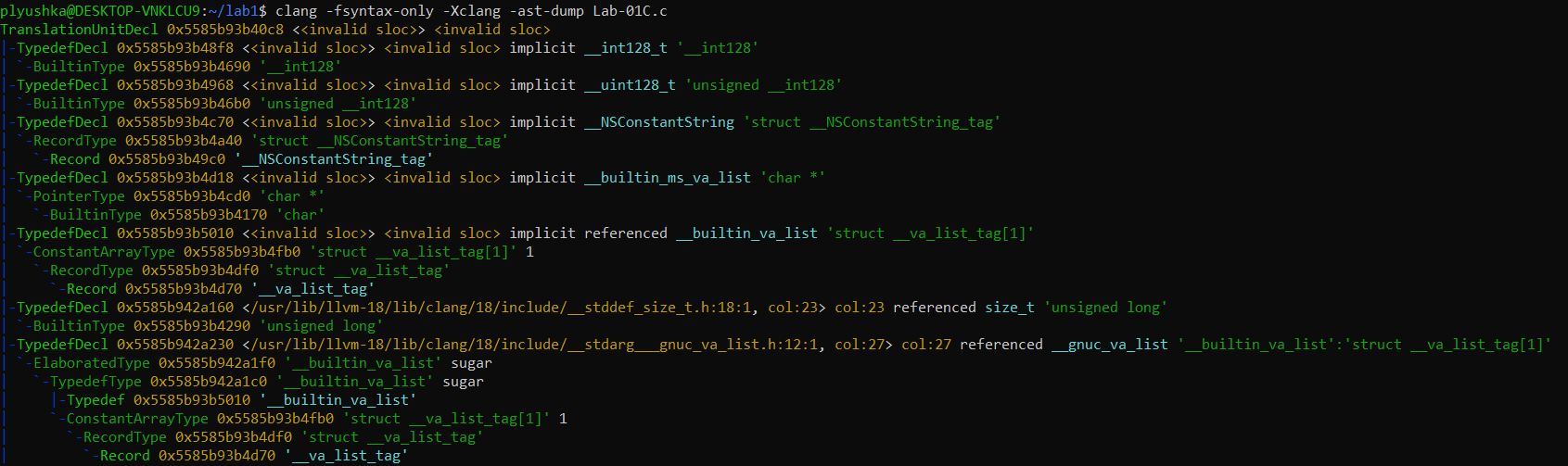
**1. Лексический анализ (токены)**

clang -fsyntax-only -Xclang -dump-tokens Lab-01C.c



**2. Синтаксический анализ (AST)**

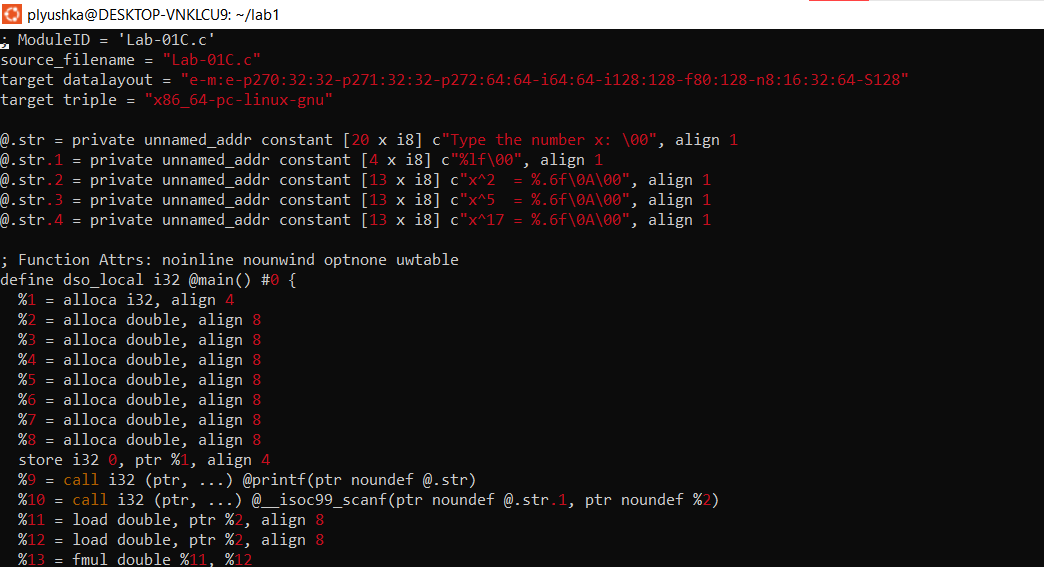
clang -fsyntax-only -Xclang -ast-dump Lab-01C.c



**3. Генерация промежуточного кода (LLVM IR)**

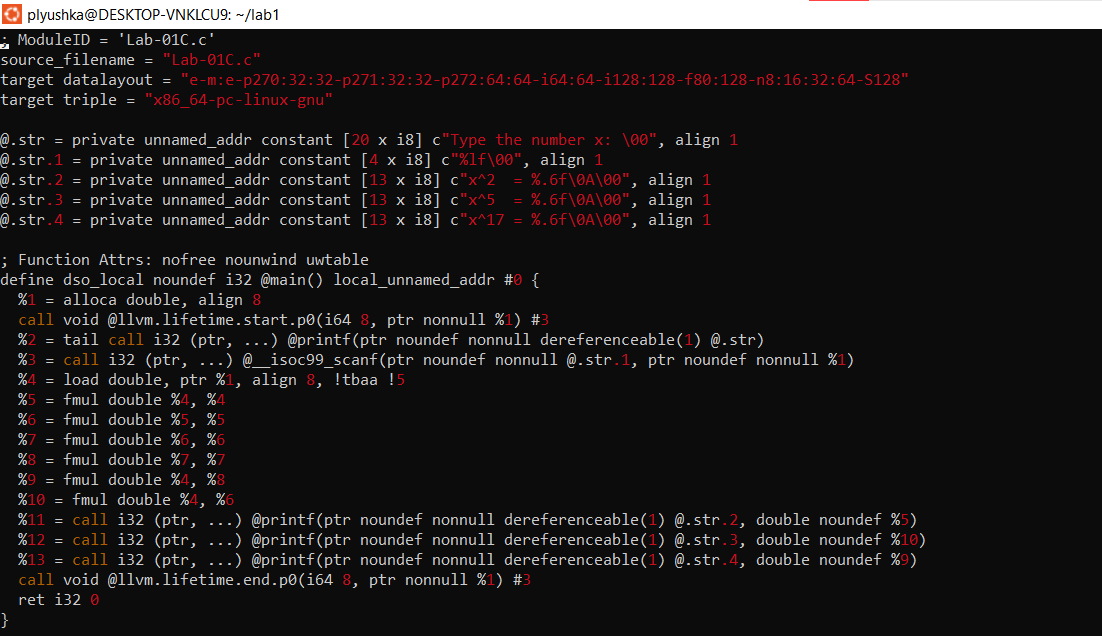
Без оптимизации

clang -S -emit-llvm Lab-01C.c -o Lab-01C.ll



С оптимизацией (-O3)

clang -S -O3 -emit-llvm Lab-01C.c -o Lab-01C\_opt.ll

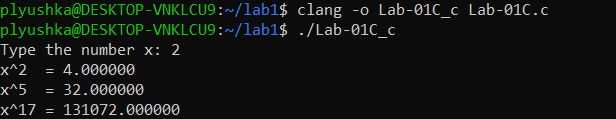


1. **Компиляция в объектный файл**

clang -c Lab-01C.c -o Lab-01C.o

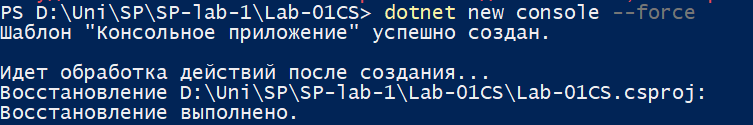


**5. Компоновка в исполняемый файл**

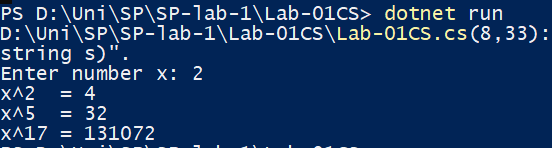


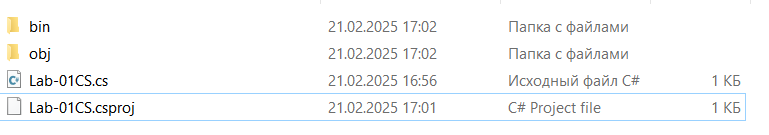
**Процесс компиляции приложения на C#**

**1. Создаём .NET-проект (консольное приложение)**

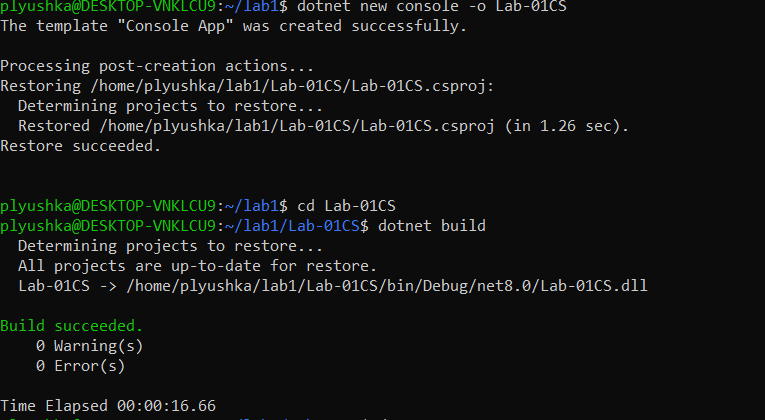


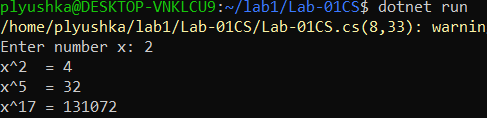
**2. Компиляция и запуск**





**Процесс компиляции приложения на C# (Linux)**





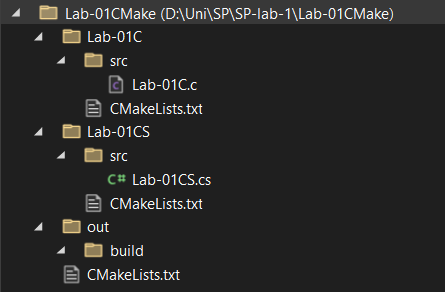
**CMake**

При использовании CMake должно быть выполнено следующее:

1. Проект для сборки должен включать в себя два приложения на C и C#;
2. На ОС семейства Linux/macOS проект С# должен игнорироваться при сборке (должно быть прописано в виде условия);
3. Сборка должна быть «out-of-source»;
4. Для Windows использовать генератор «Visual Studio», для Linux – «Unix Makefiles», для macOS – «XCode»;
5. В конфигурации CMake (CMakeLists.txt) должны быть использованы переменные для определения текущей ОС, а также вывод сообщения в логи CMake сообщения вида «Hello, I am [название ОС]!».

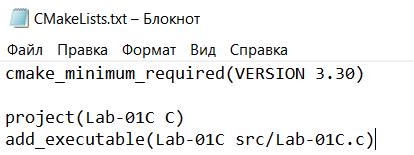
**Сборка проекта**

1. **Подготовка структуры проекта. Заполнение файлов Lab-01c.c и Lab-01CS.cs**

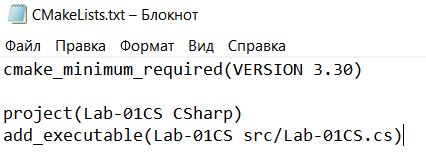


1. **Настройка файлов конфигурации.**

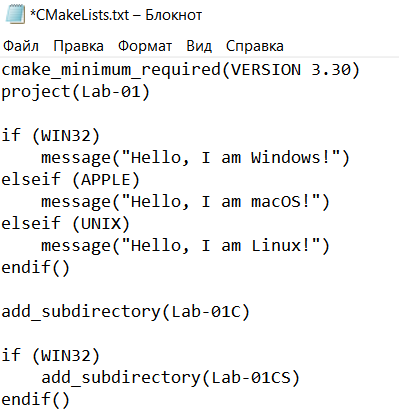
**Lab-01C/CMakeLists.txt**



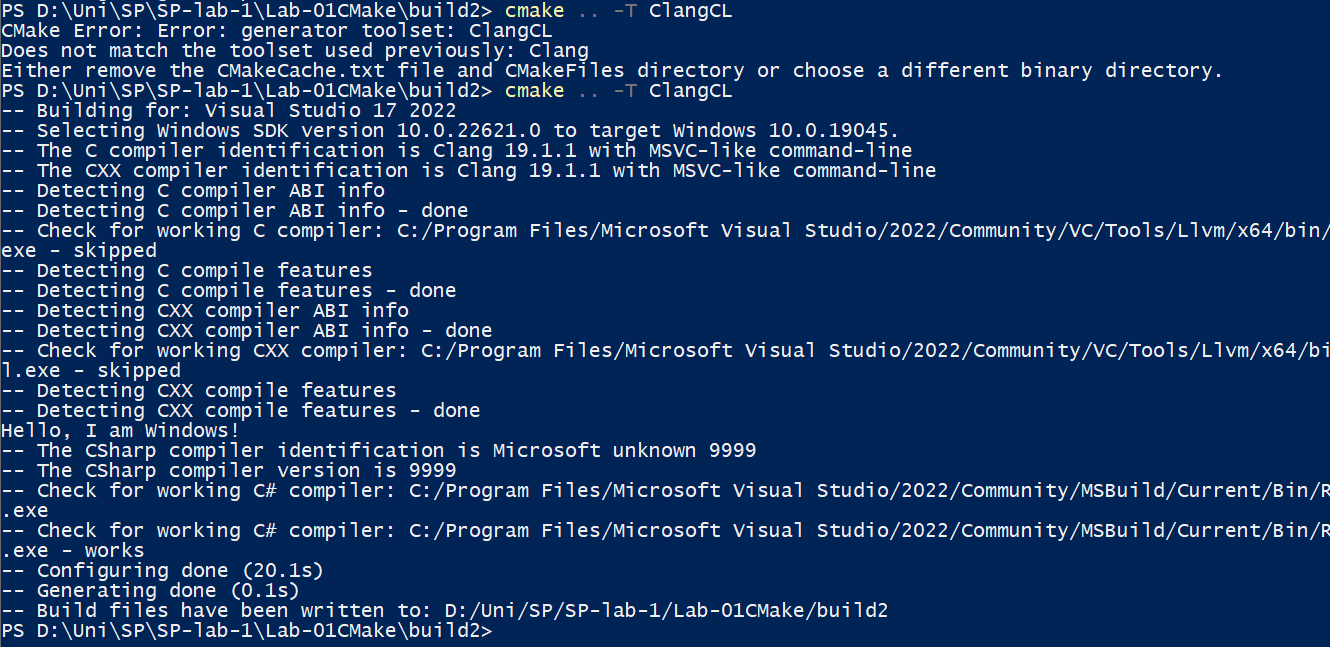
**Lab-01CS/CMakeLists.txt**

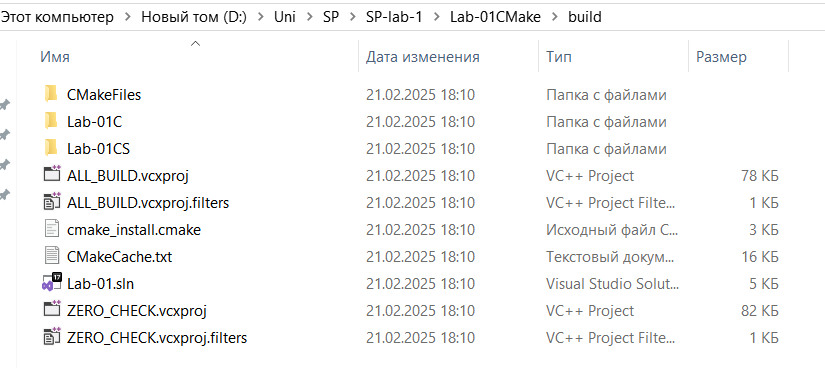


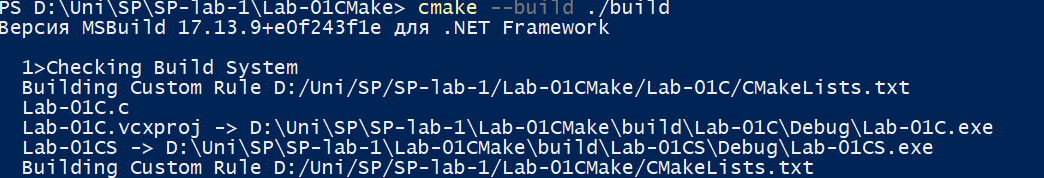
**Глобальный CMakeLists.txt**

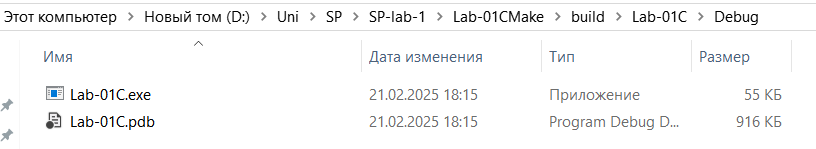


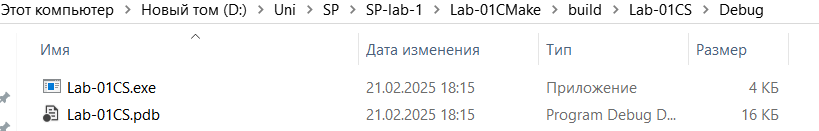
1. **Сборка проекта с CMake**







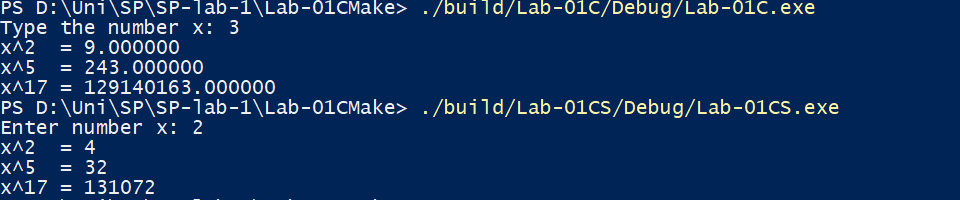




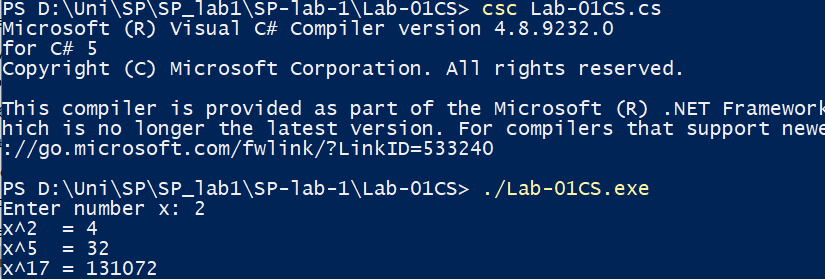
1. **Запуск приложений**

./build/Lab-01C/Debug/Lab-01C.exe

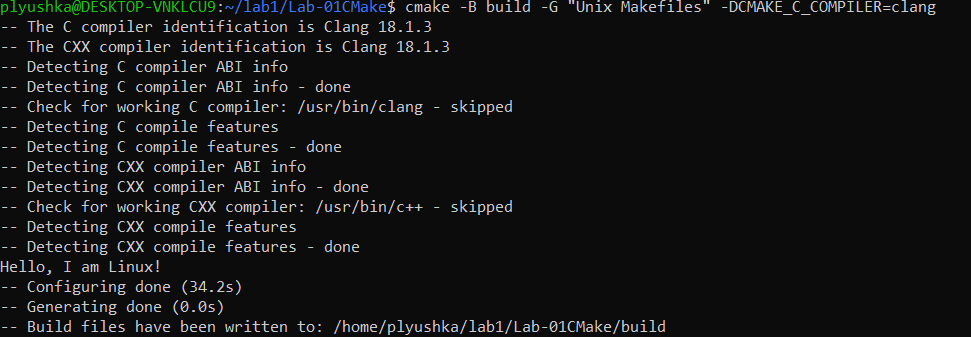
./build/Lab-01CS/Debug/Lab-01CS.exe



**Компиляция с CSC**



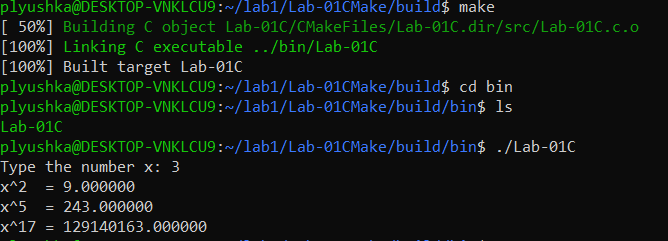
**Сборка проекта на Linux (Ubuntu)**



-B build — указывает каталог для сборки.

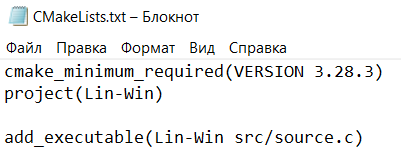
-G "Unix Makefiles" — выбирает генератор для Unix-систем.

-DCMAKE\_C\_COMPILER=clang — указывает компилятор C для CMake

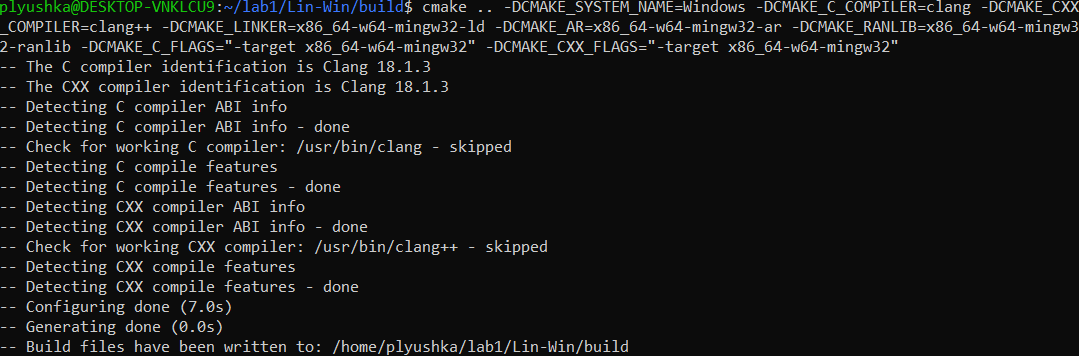


Продемонстрировать кросс-компиляцию проекта (в любом направлении: Windows->Linux/macOS или Linux/macOS->Windows).





cmake .. -DCMAKE\_SYSTEM\_NAME=Windows -DCMAKE\_C\_COMPILER=clang -DCMAKE\_CXX\_COMPILER=clang++ -DCMAKE\_LINKER=x86\_64-w64-mingw32-ld -DCMAKE\_AR=x86\_64-w64-mingw32-ar -DCMAKE\_RANLIB=x86\_64-w64-mingw32-ranlib -DCMAKE\_C\_FLAGS="-target x86\_64-w64-mingw32" -DCMAKE\_CXX\_FLAGS="-target x86\_64-w64-mingw32"



**-DCMAKE\_SYSTEM\_NAME=Windows** Устанавливает целевую операционную систему для сборки как Windows.

**-DCMAKE\_C\_COMPILER=clang** Указывает использовать компилятор Clang для компиляции исходных файлов C**- DCMAKE\_CXX\_COMPILER=clang++** Это аналогично предыдущему флагу, но для файлов на C++.

**-DCMAKE\_LINKER=x86\_64-w64-mingw32-ld** Устанавливает MinGW линковщик (ld) как компоновщик для финальной сборки.

**-DCMAKE\_AR=x86\_64-w64-mingw32-ar** Указывает на использование инструмента ar из MinGW для создания архивов. ar используется для создания статических библиотек (архивов), которые затем могут быть использованы в сборке.

**-DCMAKE\_RANLIB=x86\_64-w64-mingw32-ranlib** Устанавливает использование инструмента ranlib из MinGW. Он используется для создания индексов в статических библиотеках, чтобы ускорить доступ к их символам.

**-DCMAKE\_C\_FLAGS="-target x86\_64-w64-mingw32"** Устанавливает дополнительные флаги для компилятора C. В данном случае используется флаг -target, который указывает, что цель сборки — архитектура x86\_64 с платформой MinGW для Windows. Это сообщает компилятору, что нужно генерировать код для Windows, даже если компиляция происходит на другой операционной системе.

**-DCMAKE\_CXX\_FLAGS="-target x86\_64-w64-mingw32"** Точно такой же флаг, как и для C, но для компилятора C++.

