**Задание 1.**

1. Разработайте в Visual Studio программу **HelloWorld**, убедитесь в ее работоспособности.

2. Найдите в папке проекта созданный объектный модуль. В какой папке он находится?

компиляция исходного кода – трансляция исходного кода, написанного на одном языке программирования, в исходный код на другом языке. В результате компиляции создается файл с расширением **obj** – объектный модуль программы

 Содержание объектного файла — в сущности две вещи:

* *код*, соответствующий [определению](https://habr.com/ru/post/150327/#definition) функции в C файле
* *данные*, соответствующие [определению](https://habr.com/ru/post/150327/#definition) **глобальных** переменных в C файле (для инициализированных глобальных переменных начальное значение переменной тоже должно быть сохранено в объектном файле).

LP\_Lab01\LP\_Lab01\Debug

3. Внесите изменения в текст программы, чтобы в нем содержались ошибки. Как система программирования сообщает об ошибках?

Окно «список ошибок»

Код ошибки>>описание>>проект>>файл>>строка>>**состояние подавления**

4. Выполните сборку проекта. После успешной сборки найдите в папке проекта исполняемый модуль.

файл с исходным кодом (module.cpp) включает в себя определения функций, а также определения глобальных переменных и констант (если они есть);

5. Создайте новый проект консольного приложения.

Приложение по введенной дате в формате ДДММГГГГ должно вызывать функции:

− для определения, является ли год ГГГГ високосным;

− для вычисления порядкового номера дня в году.

Результаты выполнения вывести в консоль.

6. Скомпилируйте текст программы. Найдите в папке проекта созданный объектный модуль. В какой папке он находится?

LP\_Lab01\LP\_Lab01\Debug

7. Внесите изменения в текст программы, чтобы в нем содержались ошибки. Посмотрите, как система программирования сообщает об ошибках.

8. Выполните сборку проекта. После успешной сборки найдите в папке проекта исполняемый модуль.

9. Запустите программу на исполнение несколько раз с различными входными (введенными) данными.

10. Установите конфигурацию проекта Release и снова выполните полную сборку проекта. Откройте папку проекта. Какие изменения в ней произошли? Сравните размер отладочной и конечной версии исполняемого модуля. Объясните их различия.

11. Переключитесь в отладочную конфигурацию, установите точки останова и выполните отладочный запуск программы.

Просмотрите значение локальных переменных на момент останова.

Измените значение какой-либо переменной, присвоив ей другое корректное значение. Убедитесь, что программа будет при вычислениях использовать новое значение.

Выполните всю программу в пошаговом режиме два раза. Один раз – с трассировкой содержимого функции, второй раз – выполнив функцию в автоматическом режиме.

12. Модифицируйте код функции таким образом, чтобы у пользователя не было возможности ввести некорректную дату

**Задание 2.**

1. Используйте при выполнении лабораторной работы материал лекции 1.

2. Разработайте программу HelloWorld, убедитесь в ее работоспособности.

3. Перестройте проект. Проанализируйте раздел проекта **Внешние зависимости**. Объясните содержимое этой папки.

4. Перестройте проект. Проанализируйте директории проекта. В поддиректории **Debug** найдите файлы с расширением **obj**.

5. Исследуйте свойства проекта связанные с параметрами компилятора C++. Установите параметры компилятора:

− **Местоположение листинга ASM (/Fa)** в значение **$(IntDir);**

− **Файл ассемблерного кода** в значение **/FAcs**.

Ознакомьтесь с разделом **Командная строка**.

6. Перестройте проект. Проанализируйте ASM-листинг. Найдите в листинге ASM-представление операторов C++.

7. Ознакомьтесь с параметрами компилятора https://msdn.microsoft.com/ru-ru/library/fwkeyyhe(v=vs.110).aspx

8. Исследуйте свойства проекта связанные с параметрами компоновки. Ознакомьтесь с разделом **Командная строка**.

9. Ознакомьтесь с параметрами компоновщика https://msdn.microsoft.com/en-us/library/y0zzbyt4(v=vs.110).aspx

10. В директориях проекта найдите файл с расширением **log** (файл журнала построения). Проанализируйте и объясните его содержимое.

11. Запустите консоль **Командная строка разработчика VS201х**.

Выполните команду **SET**. Проанализируйте значения переменных окружения.

12. Создайте в корне диска **D** (или другого диска) директорий. Скопируйте из директориев проекта **HelloWorld** в созданный директорий файлы с расширением **h** и **cpp**. Выполните компиляцию и компоновку файлов в командной строке. Убедитесь в работоспособности сформированного исполняемого файла при компоновке.

**Вопросы:**

- перечислите расширения исходных файлов проекта С++;

- перечислите этапы обработки исходного кода программы;

- в результате какого этапа образуются файлы с расширением **obj**? Что в этих файлах содержится?

Сборка (build) состоит из 3 основных этапов:

на этапе препроцессинга (preprocessing) директивы препроцессора (например, #include) заменяются содержимым указанного в них заголовочного файла, в результате файл с исходным кодом дополняется прототипами указанных там функций и объявлениями глобальных переменных – за счет этого в файле можно вызывать указанные функции (прототип впереди) и использовать глобальные переменные (объявление впереди). Препроцессор может создавать на диске временные файлы, которые, однако, удаляются после окончания сборки

на этапе компиляции (compiling) для каждого исходного файла составляются таблицы определенных в нем функций и глобальных переменных, все определенные функции переводятся на машинный язык (при переводе на машинный язык могут выявляться ошибки компиляции). Результатом работы компилятора являются файлы module.obj (объектные файлы) для каждого использованного в программе модуля

- в результате какого этапа образуются файл с расширением **exe**?

на этапе связывания (linking) происходит привязка всех используемых (вызванных) функций и глобальных переменных к той таблице, в которой они определены; если определения не обнаружены ни в одной таблице, происходит ошибка связывания (unresolved external symbol …). Результатом связывания является файл program.exe (для MVS имя совпадает с именем проекта) – исполняемый файл

- объясните, каким образом программа **cl** определяет местоположение файлов, указанных в папке **Внешние зависимости** (с расширением **h**) проекта при сокращенной записи команды **cl**?

Прототипы функций находятся в заголовочных файлах (iostream, math.h, stdio.h, string.h и т.п.)

- объясните, каким образом программа **link** определяет местоположение файлов c расширением **lib** при сокращенной записи команды **link**?

Объектные файлы с уже переведенными на машинный язык определениями функций заранее собраны в библиотеки (файлы с расширением lib или dll)

определения из статических библиотек (расширение lib) на этапе связывания добавляются в исполняемый файл программы

определения из динамических библиотек (расширение dll) в исполняемый файл не добавляются; вместо этого в ссылке на соответствующую функцию указывается, что она находится в динамической библиотеке, и при ее вызове происходит обращение к библиотеке

поэтому, статические библиотеки нужны только на этапе связывания, а динамические – и на этапе исполнения программы

LINK.exe связывает объектные файлы и библиотеки Common Object File Format (COFF) для создания исполняемого файла (.exe) или динамически подключаемой библиотеки (DLL)

Линкер - это то, что создает окончательный вывод компиляции из объектных файлов, созданных компилятором. Эти выходные данные могут быть либо общей (или динамической) библиотекой (и, хотя имя схоже, они не имеют много общего со статическими библиотеками, упомянутыми ранее), либо исполняемым файлом.

Он связывает все объектные файлы, заменяя ссылки на неопределенные символы правильными адресами. Каждый из этих символов может быть определен в других объектных файлах или в библиотеках. Если они определены в библиотеках, отличных от стандартной библиотеки, вам необходимо сообщить о них компоновщику.

На этом этапе наиболее распространенными ошибками являются пропущенные определения или дубликаты определений. Первое означает, что либо определения не существуют (т.е. они не записаны), либо объектные файлы или библиотеки, в которых они находятся, не были переданы компоновщику. Последнее очевидно: один и тот же символ был определен в двух разных объектных файлах или библиотеках.