**ПЛАН ЗАНЯТИЯ**

**Дисциплина:** МДК.01.04 Системное программирование

**Преподаватель:** Галузин А.Б.

**Курс: 4**

**Группа:** П-40

**Специальность:** 09.02.07 Информационные системы и программирование

**Дата:** 25.09.25

**Время проведения:** 1 пара

**Тема:** Настройка проекта сборки в Visual Studio 2022.

**Цель занятия:**

**дидактическая:** изучить особенности разработки проектов в Masm и Visual Studio

**развивающая**: развивать абстрактное мышление, логику

**Вид занятия** лекция

**Литература**

Юров В.И. Assembler. Учебник для вузов. - 2-ое изд. – СПб.: Питер, 2003, стр. 128.

**Интернет-ресурсы:**

https://metanit.com/assembler/tutorial/1.4.php

Создание проекта в Visual Studio с ассемблерным модулем ttps://rutube.ru/video/baf45c7b660bf14335f65a7675b84aae/?r=plwd

**ЗАДАНИЕ**: законспектировать лекцию с учетом контрольных вопросов.

**Практическая работа № 5.2**

**Тема: настройка проекта сборки в Visual Studio 2019 и 2022**

**Цель:** приобретение навыков работы при создании проектов MASM в Visual Studio 2022

**Теоретические сведения:**

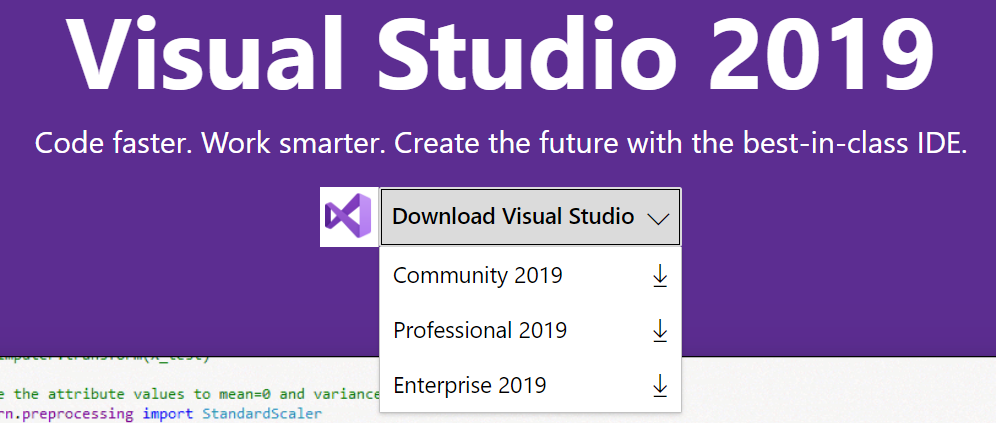
План

1. Установка Visual Studio 2019 (2022)
2. Создание проекта C++ в качестве базового проекта
3. Задайте настройку сборки как язык ассемблера
4. Создание исходного файла сборки
5. Файл со списком настроек
6. Настройка сборки в режиме выпуска
7. Сохранить как шаблон проекта
8. Приложение A: Отладка
9. Приложение B. Настройка проекта 64-битной сборки
10. Приложение C. Соглашение о вызовах Microsoft x64

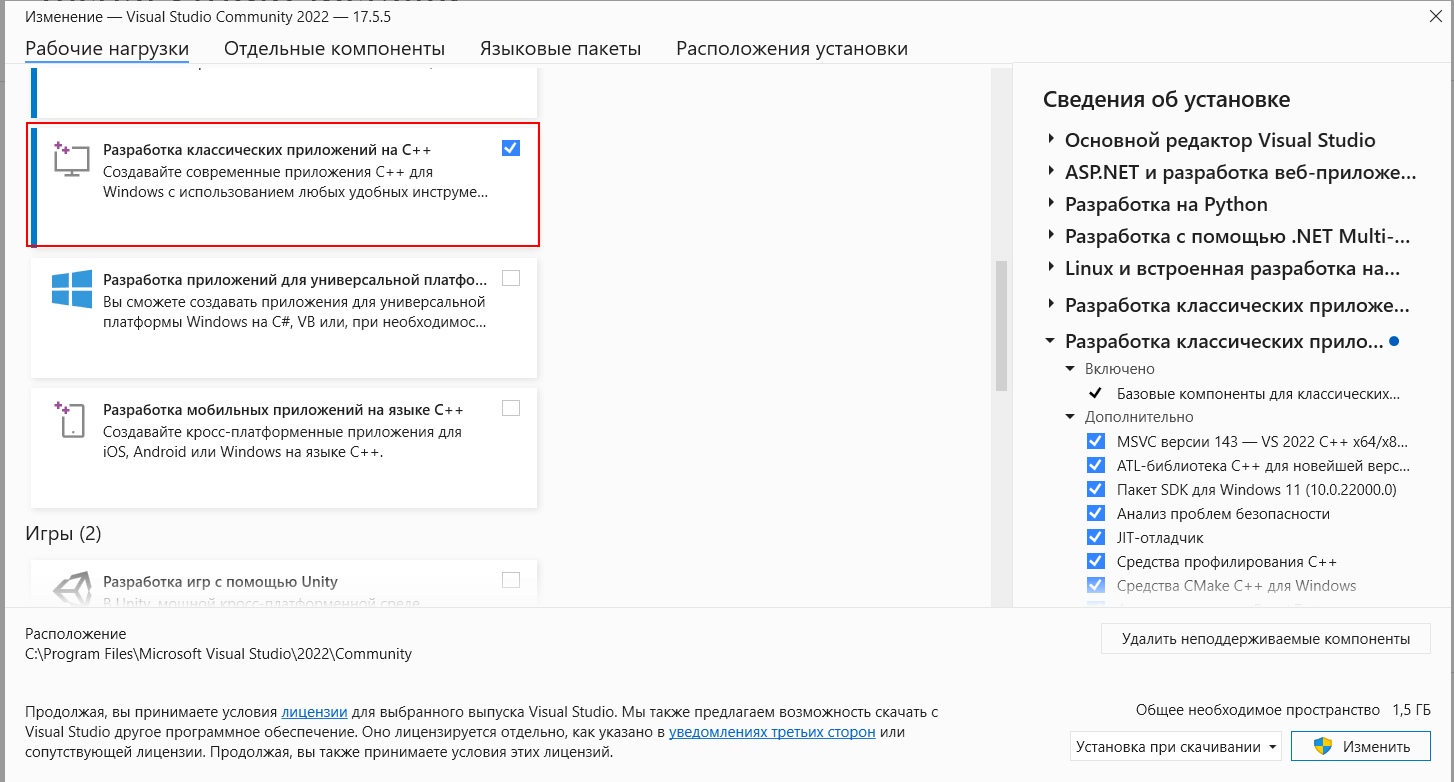
Microsoft Visual Studio поставляется со встроенным ассемблером макросов и предоставляет множество возможностей отладки (просмотр регистров и памяти, точки останова и переход). Однако не существует явного шаблона проекта сборки, из которого можно было бы выбрать. Здесь мы опишем шаги, необходимые для настройки проекта сборки в Visual Studio 2019 и 2022.

**1. Установка Visual Studio 2019 (2022)**

Visual Studio 2019 можно скачать с сайта: [https://visualstudio.microsoft.com/vs/ .](https://visualstudio.microsoft.com/vs/)   
Версия Community бесплатна.

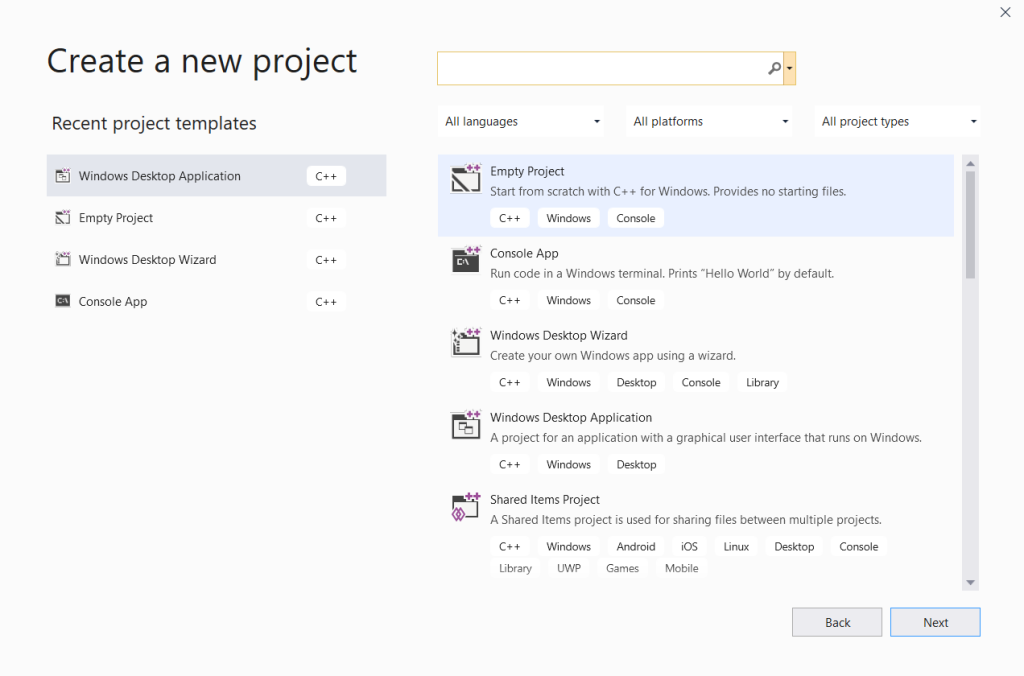
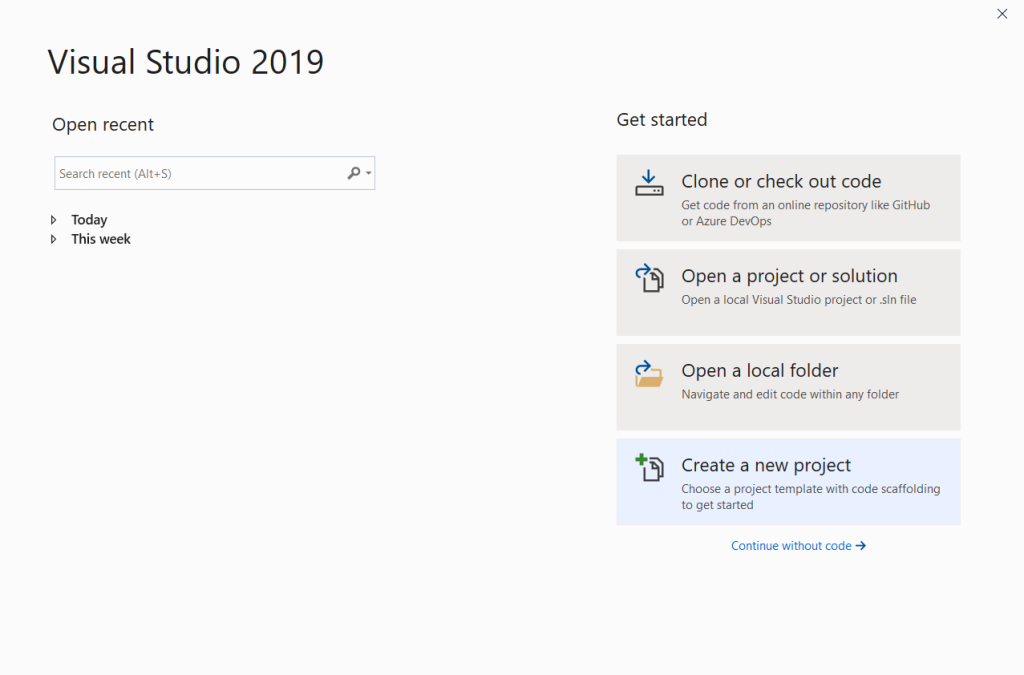


Для работы с MASM надо установить для Visual Studio инструменты разработки для C/C++.



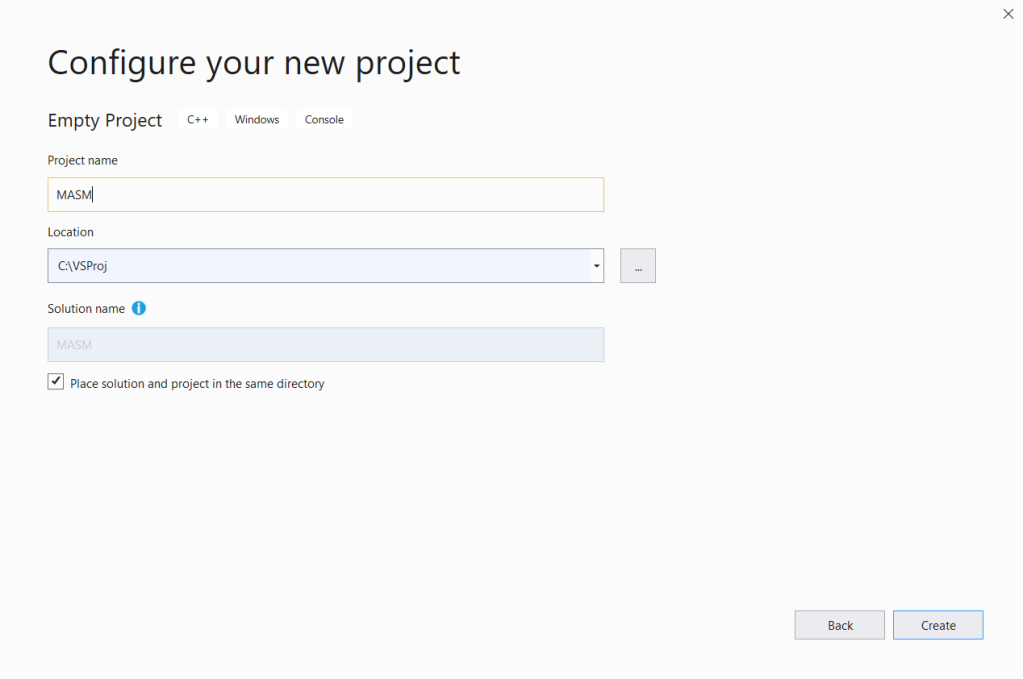
**2. Создание проекта C++ в качестве базового проекта**

Создайте пустой проект C++.



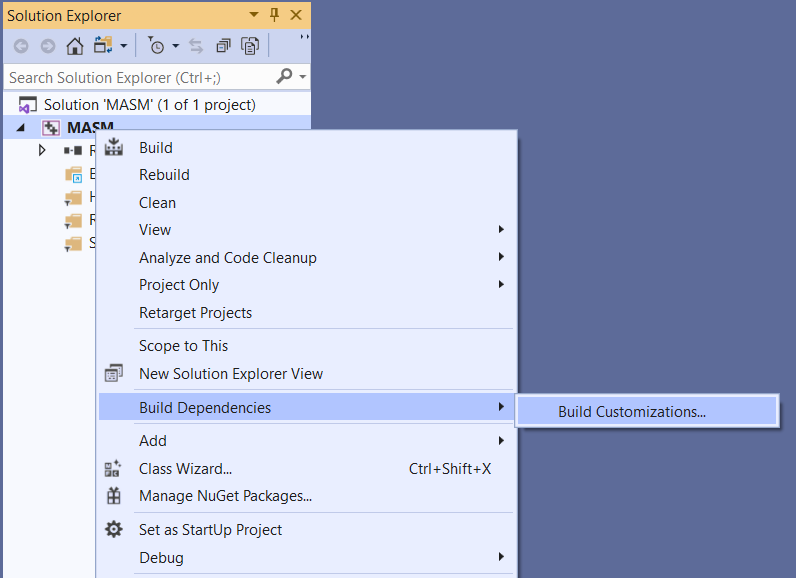
Назовём проект: «MASM». Чтобы на более позднем этапе мы могли сохранить этот проект как шаблон проекта для дальнейшего использования. Установите место сохранения в любую предпочтительную папку (я заранее создал папку C:\VSProj). Выберите «Поместить решение и проект в один каталог», чтобы предотвратить появление двух вложенных папок с одинаковым именем.

Visual Studio автоматически создаст папку с тем же именем проекта.



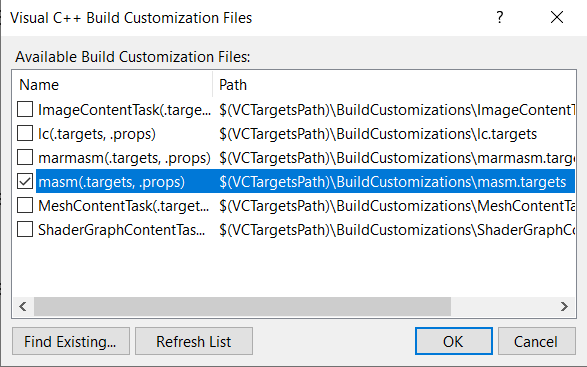
**3. Задайте настройку сборки как язык ассемблера**

Щелкните правой кнопкой мыши имя проекта > Зависимости сборки > Настройки сборки…



Выберите «masm», затем ОК.

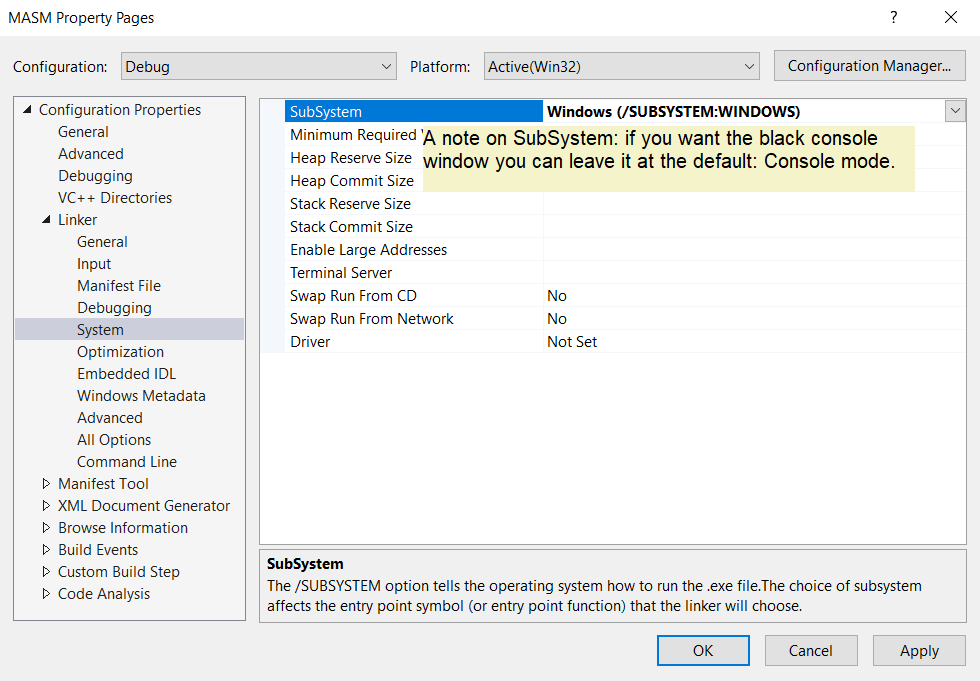
(Он не должен быть уже выбран. Если это так, вам следует перезапустить Visual Studio и повторить попытку.)



Щелкните правой кнопкой мыши **имя проекта > свойства > компоновщик > система > подсистема: Windows.**

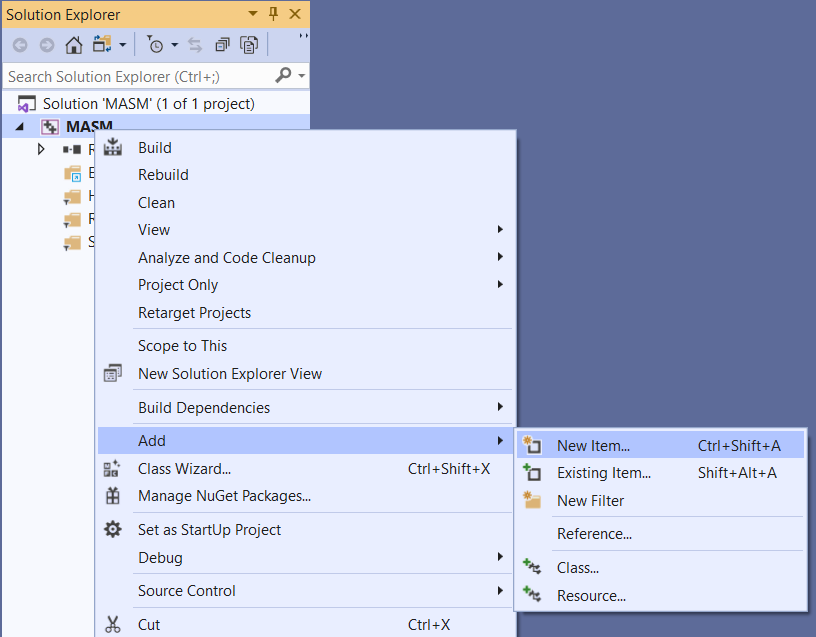
Это не позволит ассемблерной программе отображать окно консоли (черное окно текстового режима). Или, если вам нужно окно консоли, вы можете оставить его по умолчанию: **режим консоли.**

Я предпочитаю режим Windows, потому что он выглядит как родное приложение Windows. Для отладки просто используйте точку останова и окна просмотра, чтобы просматривать значения регистров и памяти в режиме реального времени. Однако, если мы хотим протестировать ввод/вывод текстового режима, нам следует использовать консольный режим.

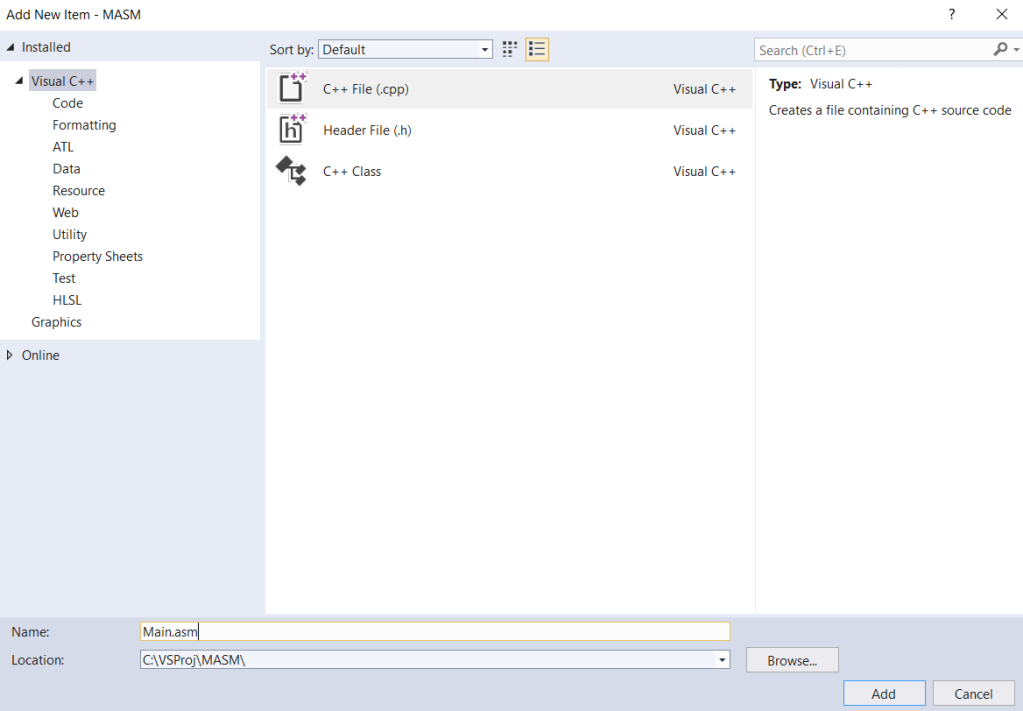


**4. Создание исходного файла сборки**

Щелкните правой кнопкой мыши имя проекта > Добавить > Новый элемент…



Назовите файл: «Main.asm». Выбранный тип элемента не имеет значения. Можно выбрать любой тип. Но важно назвать его с расширением файла «.asm».



Напишите тестовую программу. Вы можете скопировать код снизу. Я адаптировал код из замечательной книги **Кипа Ирвина «Язык ассемблера для процессоров x86»**. Я настоятельно рекомендую эту книгу всем, кто начинает изучать язык Ассемблера.

Во-первых, мы создадим 32-битный ассемблерный код. А затем в приложении Б мы создадим 64-битный ассемблерный код.

**32-битный ассемблерный код:**

.386

.model flat, stdcall

.stack 4096

ExitProcess PROTO, dwExitCode:DWORD

.data

myList DWORD 2, 3, 5, 8

.code

main PROC

mov eax, 7

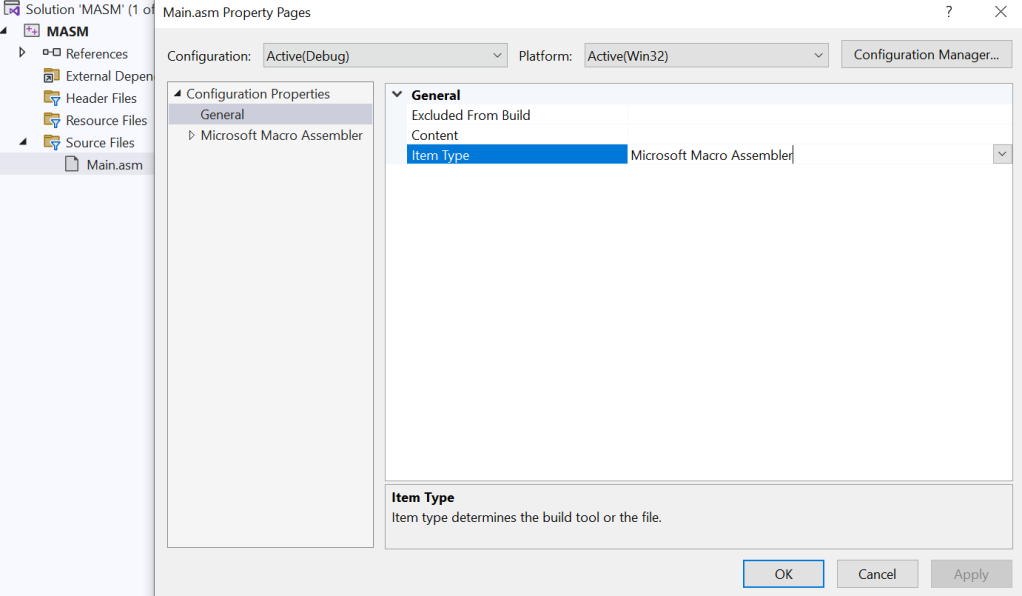
add eax, 8

INVOKE ExitProcess, eax

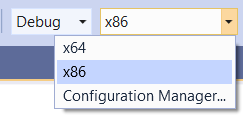
main ENDP

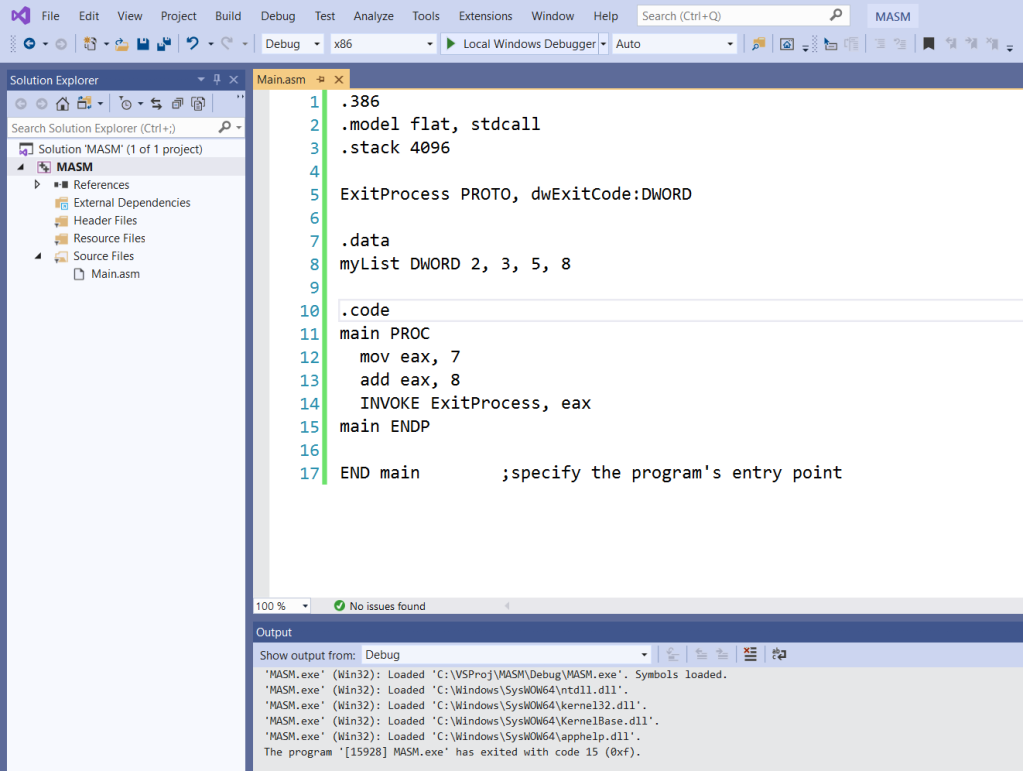
END main ;specify the program's entry point

Для Visual Studio 2022 нам нужно выбрать тип элемента исходного кода: Microsoft Macro Assembler.



Чтобы собрать 32-битную сборку, убедитесь, что в раскрывающемся списке «Платформа решения» на панели инструментов выбрано «x86».



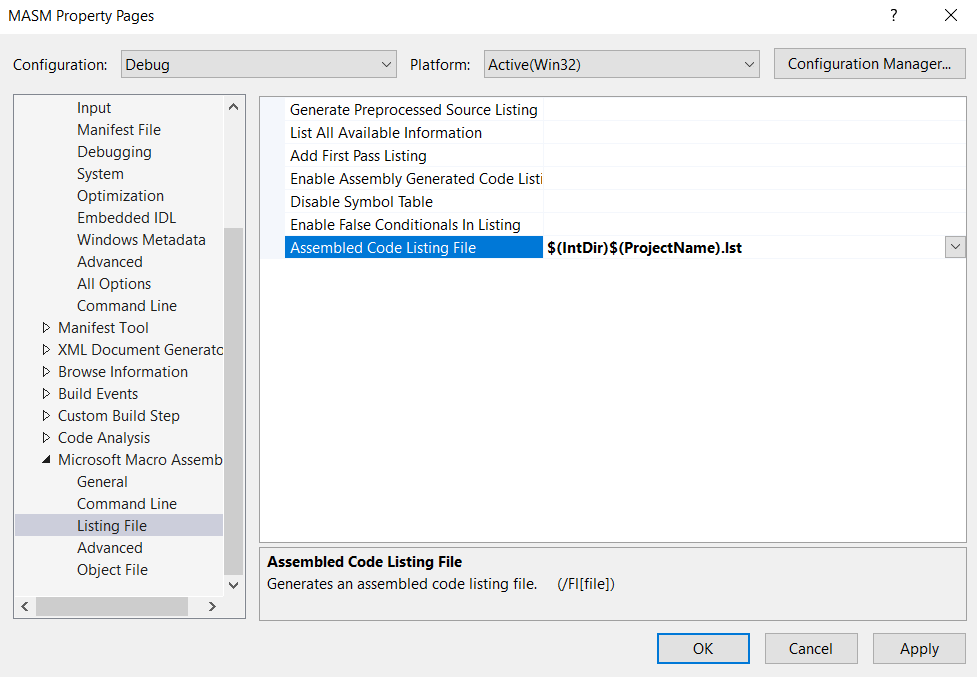


Нажмите зеленую кнопку воспроизведения: (запустите с помощью) локального отладчика Windows.   
Программа суммирует 7 с 8 и возвращает результат 15, как показано в окне вывода отладки.

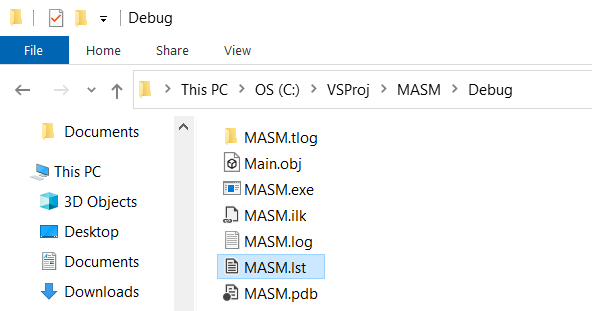
**5. Файл со списком настроек (необязательно)**

Файл листинга полезен для проверки того, что сгенерировал ассемблер. Чтобы настроить создание файла листинга, щелкните правой кнопкой мыши имя проекта > Свойства > Microsoft Macro Assembler > Файл листинга > Файл листинга собранного кода: $(IntDir)$(ProjectName).lst

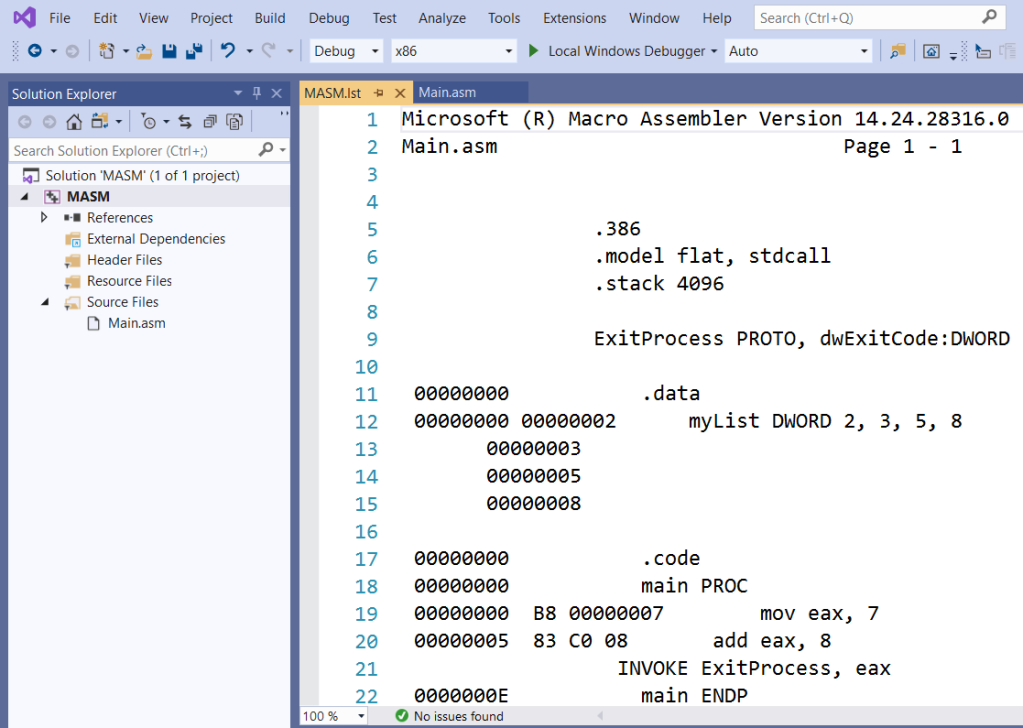
**Примечание.** Страница «Microsoft Macro Assembler» появится только в том случае, если в проекте есть файлы сборки.



Создайте (и/или запустите) проект еще раз. Вы увидите файл листинга, созданный в папке отладки проекта.



Перетащите файл списка в Visual Studio, чтобы просмотреть его. Содержимое файла листинга будет таким:



**6. Настройка сборки в режиме выпуска (необязательно)**

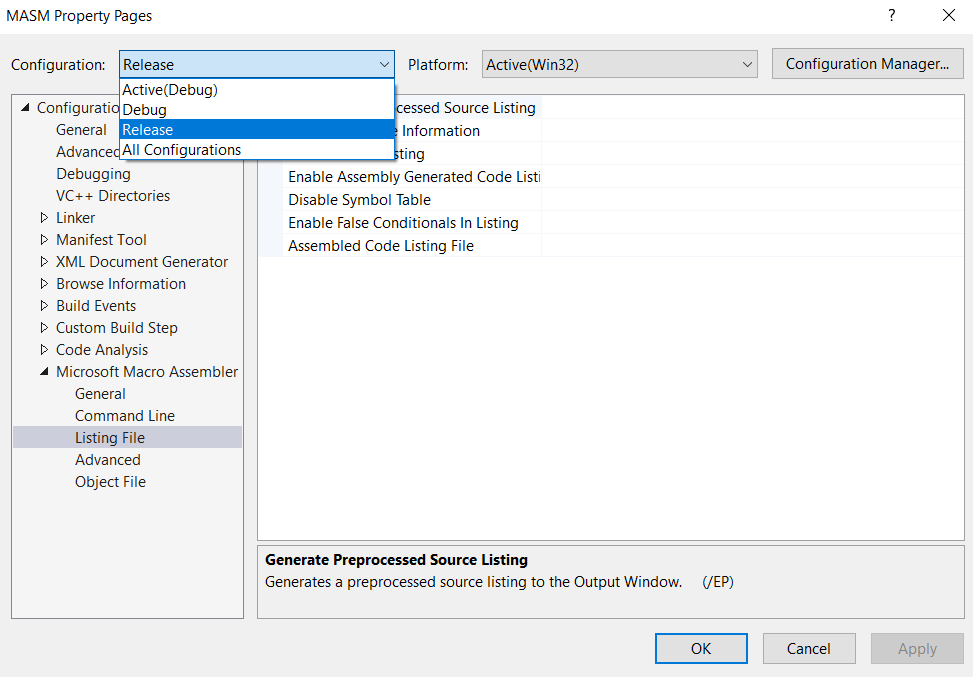
**Этот раздел можно пропустить, поскольку он довольно сложен.** Делайте это, когда вам нужно использовать режим выпуска для сборки выпуска. Или вы можете зайти в этот раздел позже, когда понадобится.

В Visual Studio существует отдельная конфигурация сборки режима выпуска. Конфигурация режима выпуска может быть установлена ​​иначе, чем конфигурация режима отладки. Например, в режиме выпуска может быть больше оптимизаций и меньше функций отладки.

Если необходим режим выпуска, мы должны сделать то же самое, что и шаги 3 и 5. Кроме того, нам необходимо настроить конфигурации безопасного обработчика исключений.

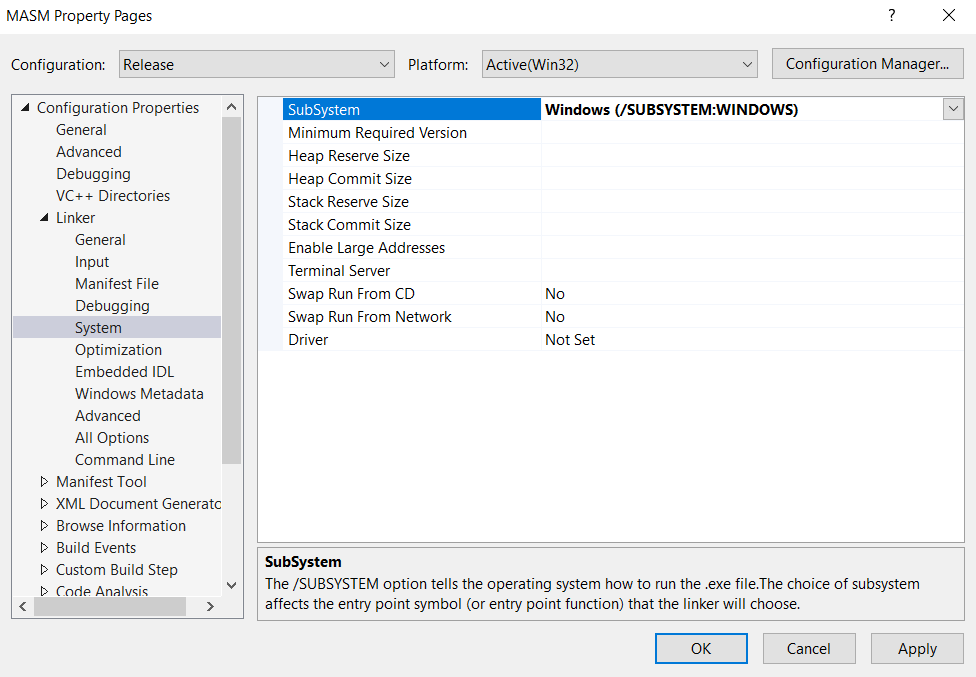
**6.1 Настройка подсистемы (Windows или консоль) и создание файла листинга**

Перейдите в свойства проекта > измените режим конфигурации на: Release.

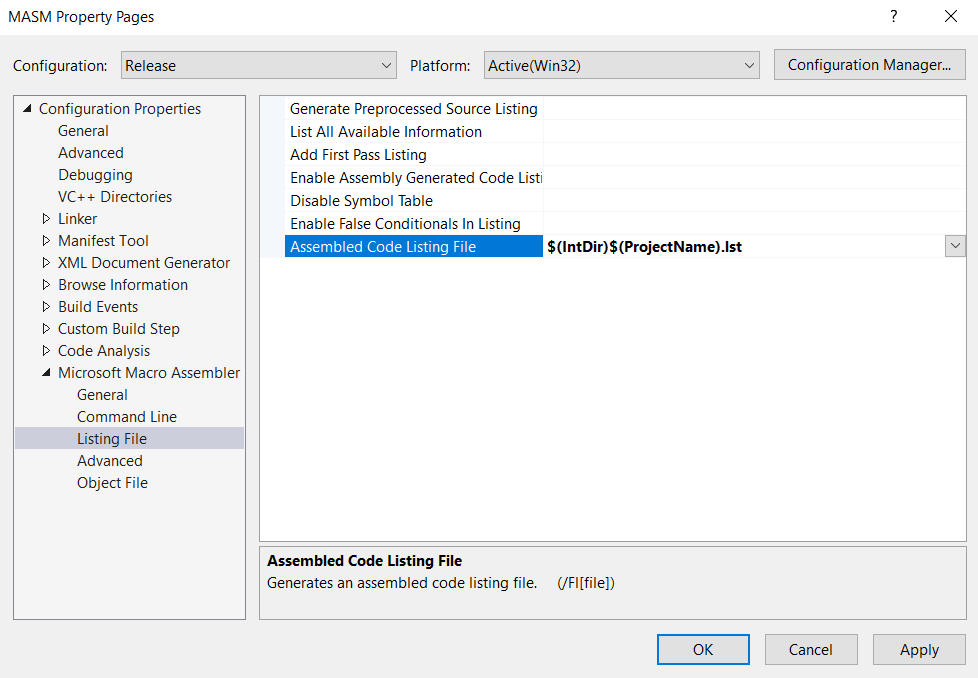


**Изменить тип приложения с консольного на оконное** (Windows или Console).

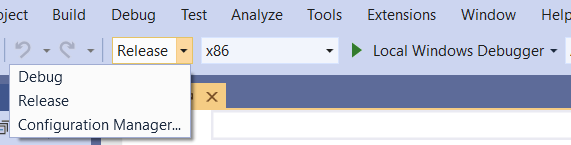
Чтобы убрать консоль (поменять тип приложения с консольного на оконное, или наоборот) необходимо обратиться к меню Свойства проекта, вызванного по правой кнопке мыши.



Настройка генерации файла листинга.

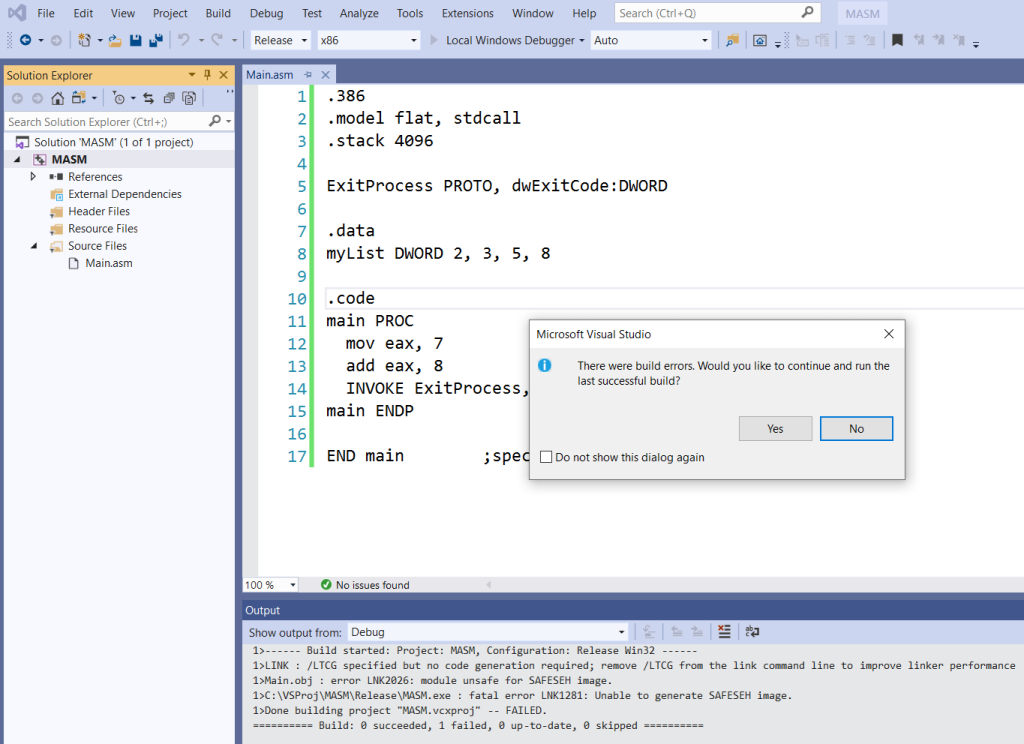


Изменение активной конфигурации на **Release**.



Попробуйте собрать и запустить программу, используя зеленую кнопку воспроизведения. Вы увидите сообщение об ошибке: **«Неустранимая ошибка: LNK1281: невозможно создать образ SAFSEH».**

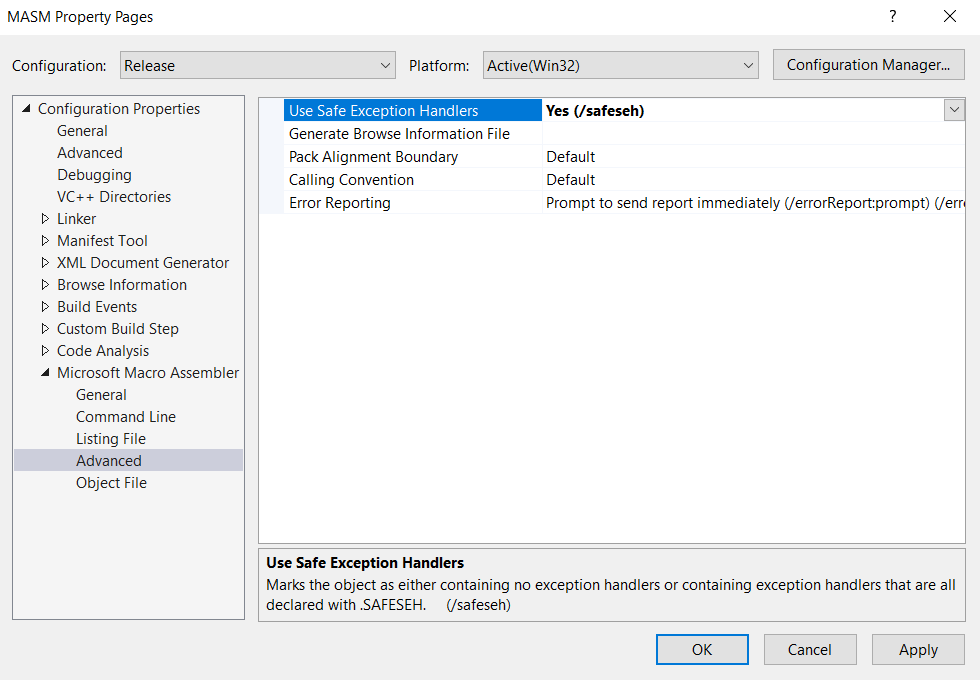
Следующий шаг 6.2 устранит эту ошибку.



**6.2 Полностью включить или отключить безопасные обработчики исключений**

В режиме сборки выпуска опция компоновщика по умолчанию заключается в том, что каждый образ (объектный файл) должен иметь безопасные обработчики исключений. Однако по умолчанию ассемблер (компилятор) не использует безопасные обработчики исключений. Эти конфликты вызывают вышеуказанную ошибку. Теперь у нас есть два варианта исправить ошибку.

**Способ 1:** включить безопасные обработчики исключений в ассемблере (компиляторе).



**Способ 2:** Или мы могли бы полностью отключить использование безопасного обработчика исключений в компоновщике.   
Компоновщик > Дополнительно > Образ имеет безопасные обработчики исключений: Нет

**Выбирайте один путь, но не оба**!

Я предпочел способ 1, потому что безопасные обработчики исключений выглядят безопаснее. Но на самом деле я не уверен на 100%.

После этого мы сможем собрать и запустить программу в режиме выпуска без каких-либо ошибок.

**Внимание при переключении между режимами «Debug» и «Release».** При изменении активной конфигурации на панели инструментов и последующем открытии страниц свойств проекта страницы свойств могут не соответствовать текущей активной конфигурации. Это может вызвать путаницу, поскольку изменение некоторых настроек проекта не влияет на текущую сборку. В этом случае убедитесь, что активная конфигурация и страницы свойств совпадают.

**7. Сохранить как шаблон проекта**

После этих долгих шагов нам, возможно, не захочется выполнять их каждый раз для создания нового проекта сборки. К счастью, Visual Studio может сохранить проект как шаблон проекта.

Сначала отредактируйте файл Main.asm, чтобы он содержал только код шаблона:

.386

.model flat, stdcall

.stack 4096

ExitProcess PROTO, dwExitCode:DWORD

.data

.code

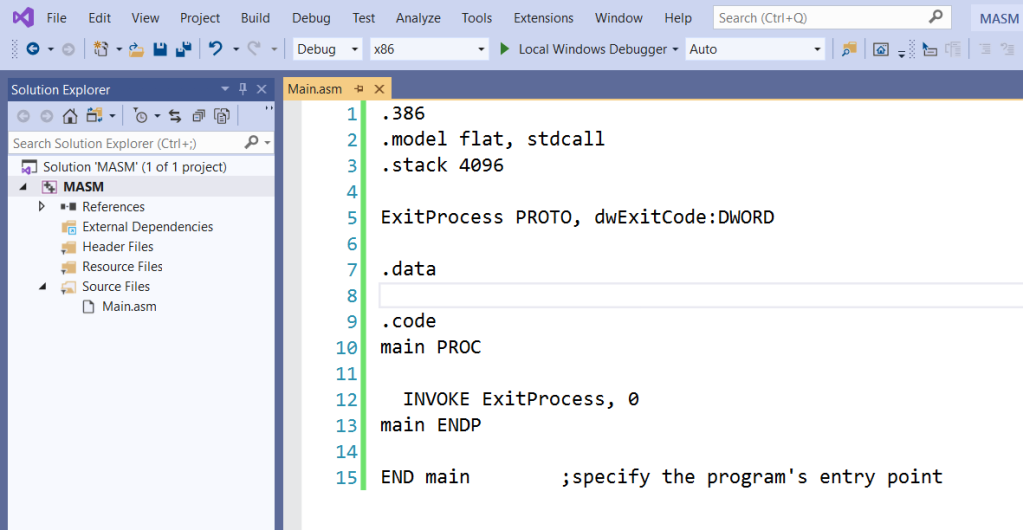
main PROC

INVOKE ExitProcess, 0

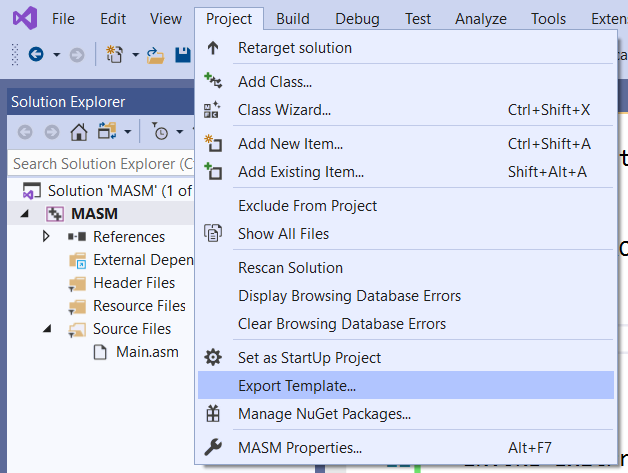
main ENDP

END main ;specify the program's entry point

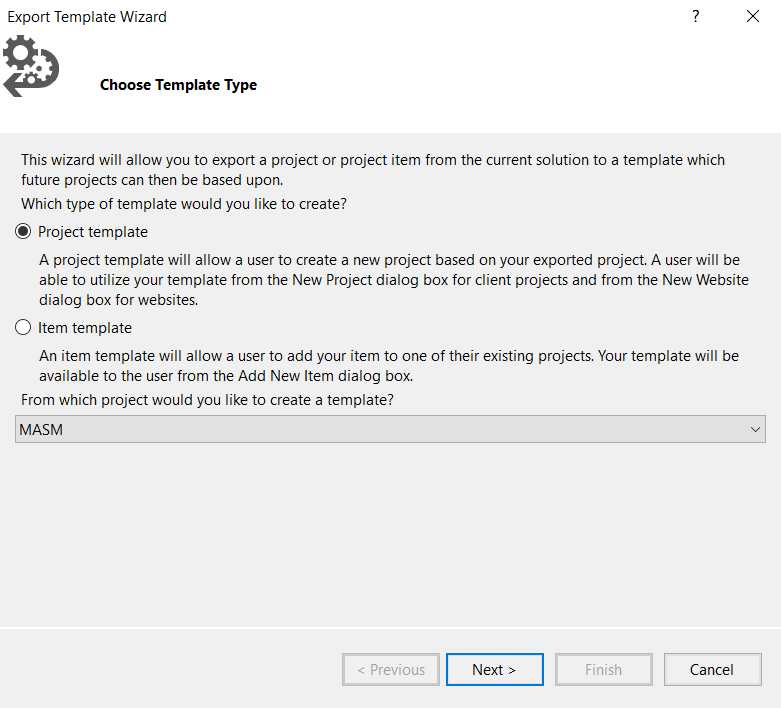
Попробуйте запустить его, чтобы убедиться в корректной работе программы.   
Если вы все еще используете режим выпуска (начиная с шага 5), вернитесь в режим отладки.



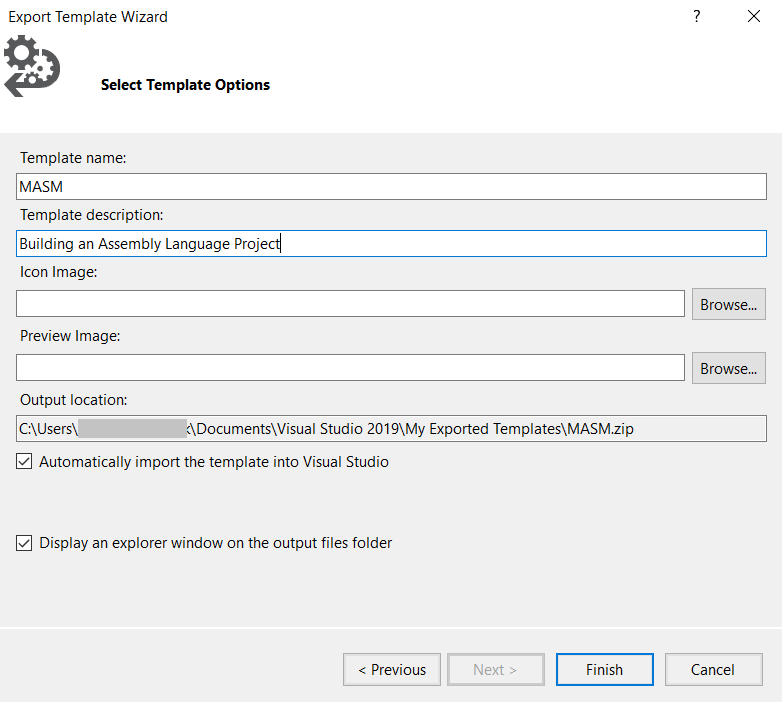
Нажмите: Проект > Экспортировать шаблон…



Выберите наш проект «MASM» из выпадающего списка.

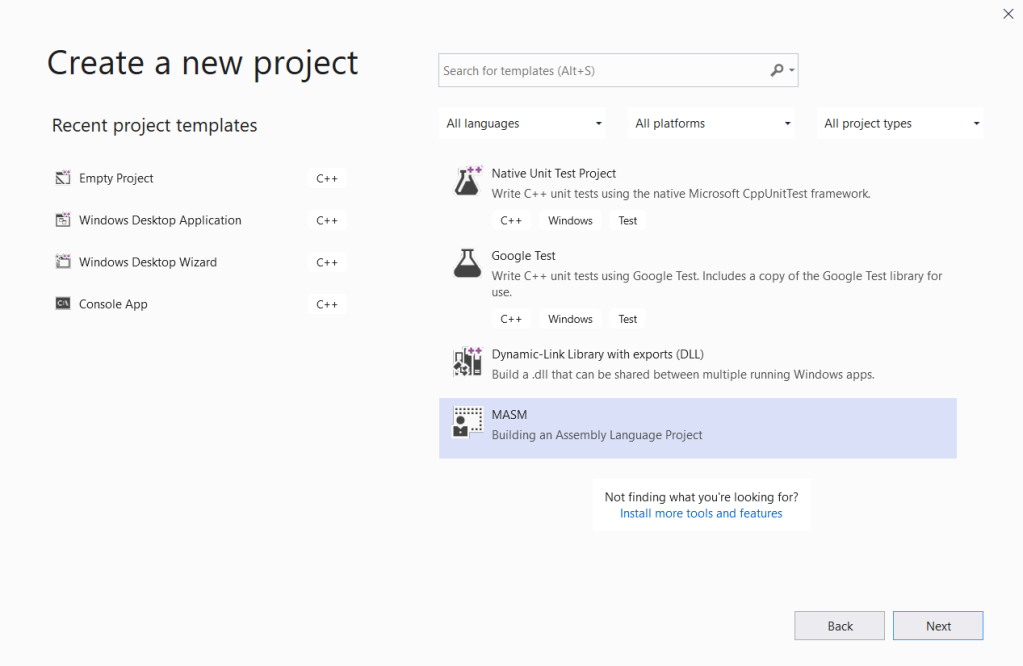


Опишите описание шаблона.



Завершено!

После этого, когда вы создаете новый проект, вы можете выбрать MASM в качестве шаблона проекта.

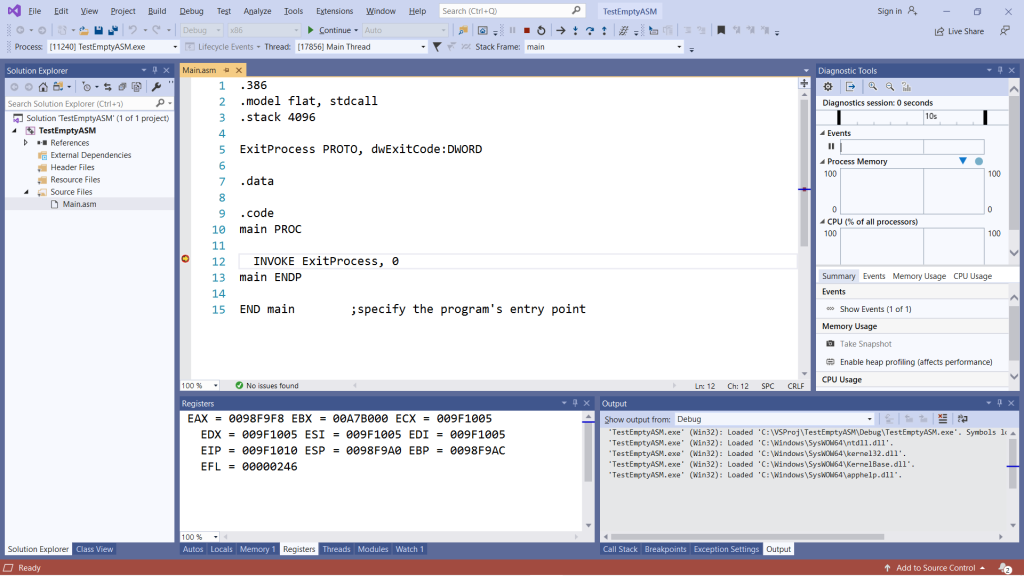


**Приложение A: Отладка**

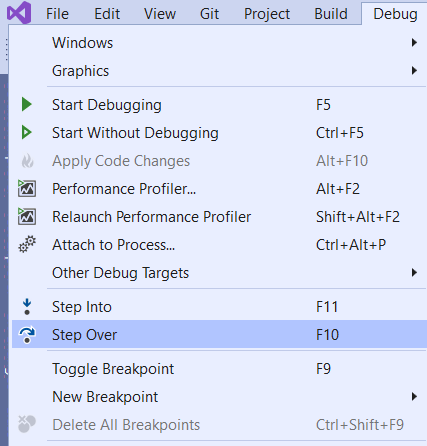
Отладка в Visual Studio проста. Чтобы установить точку останова, достаточно щелкнуть серую область перед строкой исходного кода. При запуске программа будет делать паузу перед выполнением строки точки останова.

Чтобы просмотреть регистры, выберите «Отладка» > «Windows» > «Регистры».

Чтобы просмотреть переменные, выберите «Отладка» > «Windows» > «Наблюдение». Затем введите имя переменной. Или выберите «Отладка» > «Windows» > «Память» > «Память 1». Затем введите «&varname» в поле адреса.

Пример паузы в точке останова отладчика Visual Studio для языка ассемблера

Кроме того, мы можем выполнять отладку шаг за шагом, используя «Шаг захода» и «Шаг через».

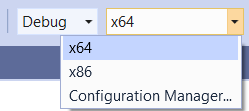


**Приложение B. Настройка проекта 64-битной сборки**

В этом разделе описаны дополнительные шаги, необходимые для настройки проекта 64-разрядной сборки. Прежде чем выполнять эти шаги, вам следует завершить предыдущие шаги.

**1. Выберите 64-битную платформу**

На панели инструментов установите для платформы решения значение «x64».



**2. Создание 64-битного исходного кода сборки**

Скопируйте приведенный ниже код, чтобы заменить старый код в main.asm:

**64-битный ассемблерный код**

ExitProcess PROTO

.data

.code

main PROC

sub rsp, 28h ; резервирование области стека в качестве области передачи параметров

mov rcx, 12345678 ; указывает код выхода

call ExitProcess

main ENDP

END

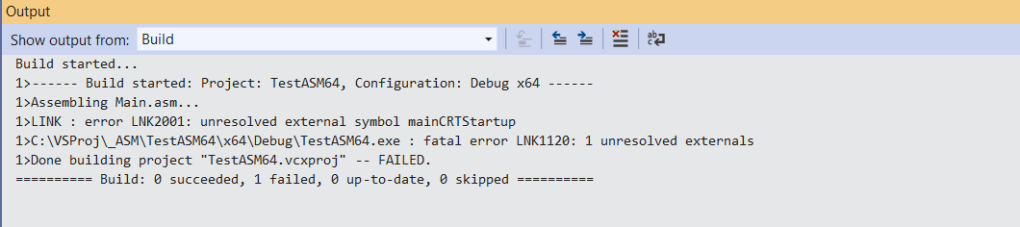
Обратите внимание, что 32-битный и 64-битный ассемблерный код во многом различаются.

В 64-битном ассемблерном коде:

* Некоторые директивы, например «.386», не используются.
* В PROTO не указан список параметров.
* В END не указан «основной» начальный символ.
* При вызове API Win32, такого как «ExitProcess», параметры будут передаваться в регистрах (rcx, rdx, r8, r9). (подробнее см. в Приложении C)

**3. Указание «главного» стартового символа (точки входа)**

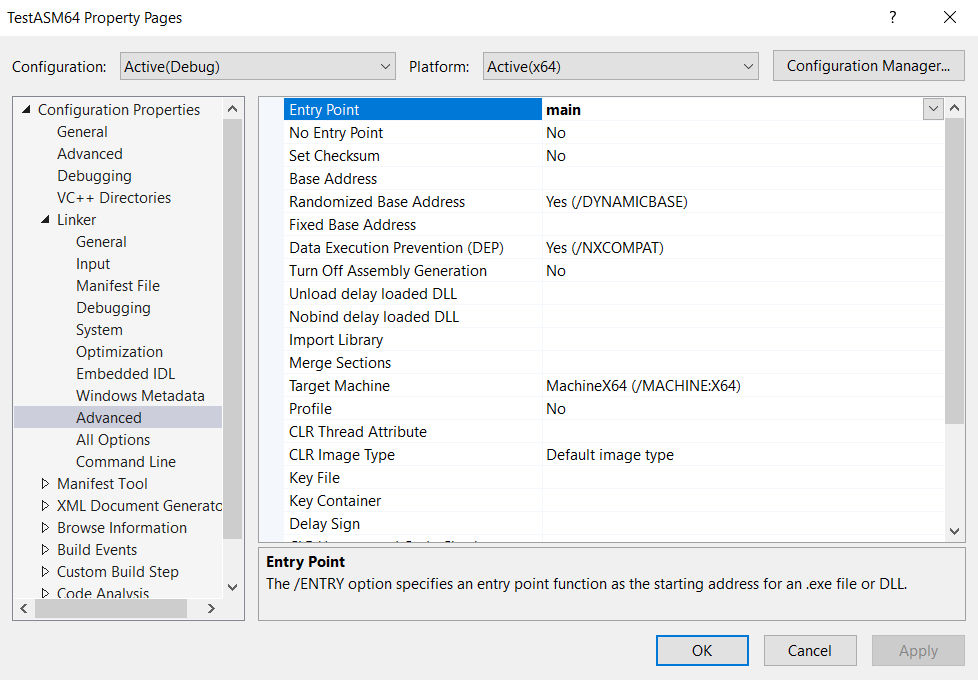
При сборке вы можете получить сообщение об ошибке: неразрешенный внешний символ mainCRTStartup.



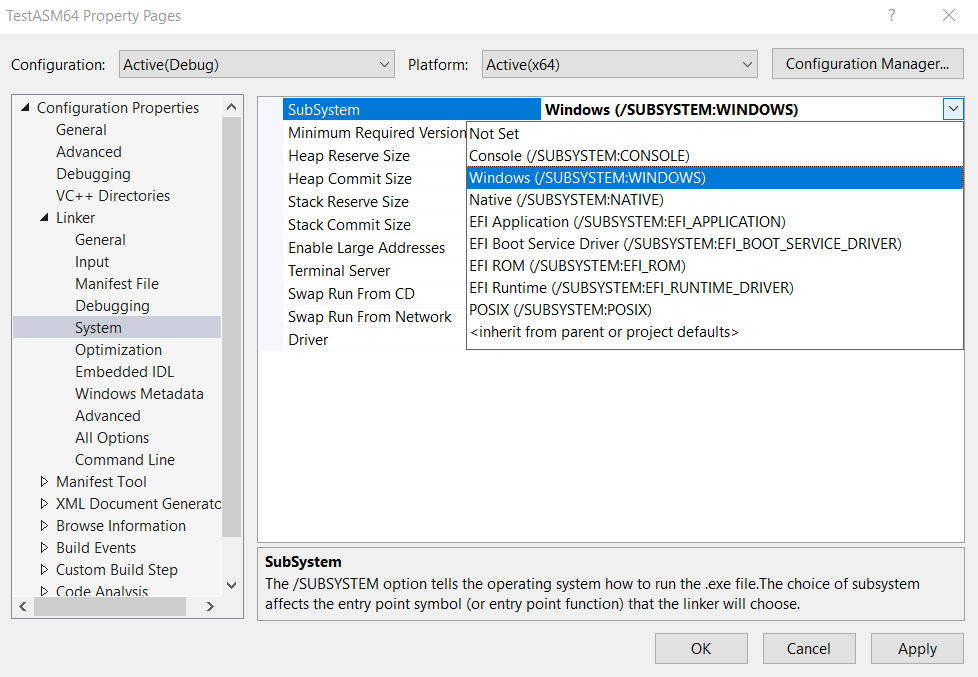
Чтобы исправить эту ошибку:

* Откройте страницу свойств проекта (см. изображение ниже), выбрав «Проект» > «Свойства».
* Затем убедитесь, что выбраны конфигурация «Активная (отладка)» и платформа «x64».
* Установите для Linker > Advance > Entry Point значение « **main** ».

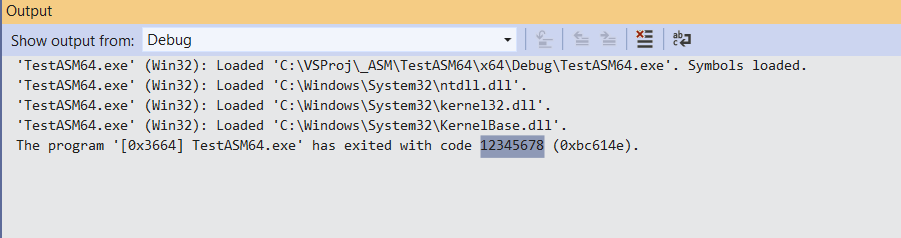
Если вы также хотите использовать сборку режима выпуска, сделайте то же самое для конфигурации: «Релиз», платформа: «x64».



Вы также можете выбрать подсистему: Консоль или Windows по своему усмотрению.



Запустите программу. Код возврата от вызова ExitProcess будет корректно отображаться в окнах вывода.



**Приложение C. Соглашение о вызовах Microsoft x64**

Microsoft определила способ, которым 64-битный ассемблерный код должен вызывать Win32 API. Эта спецификация называется [«Соглашение о вызовах Microsoft x64»](https://docs.microsoft.com/en-us/cpp/build/x64-calling-convention?view=msvc-170) . Некоторые важные моменты:

* Первые четыре аргумента передаются регистрами (rcx, rdx, r8, r9).
* Некоторая стековая память, называемая теневой областью, должна быть зарезервирована (не менее 4\*8 байт).
* Выравнивание 16-байтового указателя стека (rsp = регистр указателя стека) необходимо перед вызовом любого API Win32.

Вот причины появления этой строки в 64-битном ассемблерном коде:

sub rsp, 28h ;reserved shadow area

Нам нужно зарезервировать как минимум 32 байта (20h) для теневой области. И основной процесс всегда начинается с адреса, заканчивающегося 8h. Итак, чтобы выполнить 16-байтовое выравнивание, нам нужно добавить еще (8h) байт. Вот почему мы вычитаем rsp на (28h) байт. (стек растет вниз)

**Отображение окна сообщения**

Рассмотрим следующий код. Этот код вызывает API MessageBox Win32.



ExitProcess PROTO

MessageBoxA PROTO

.data

myText db "Hello World", 0

myCaption db "Message Box Testing", 0

.code

main PROC

sub rsp, 28h ; reserved shadow area

mov rcx, 0

lea rdx, myText

lea r8, myCaption

mov r9, 0

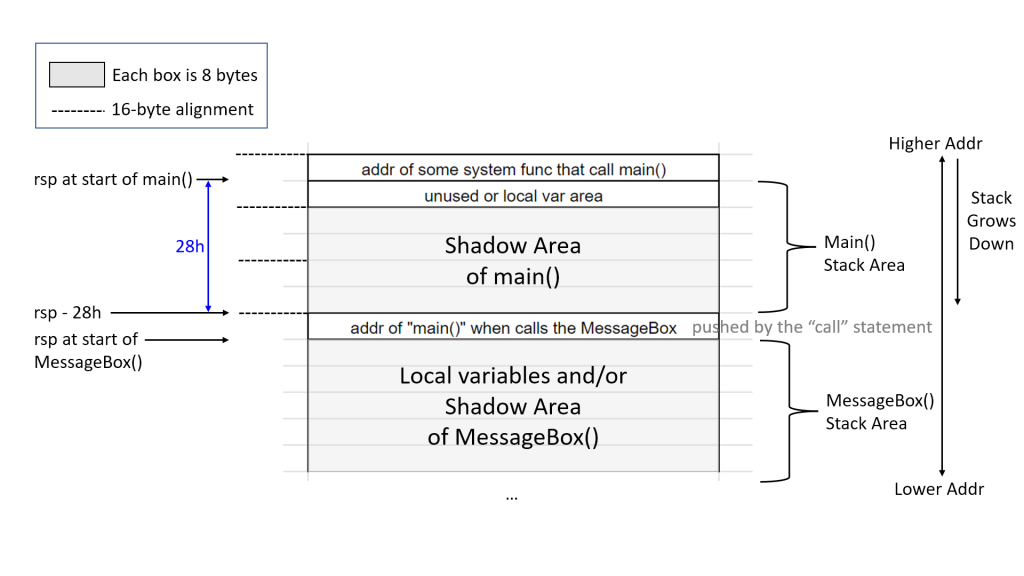
call MessageBoxA

mov rcx, 12345678 ; the exit code

call ExitProcess

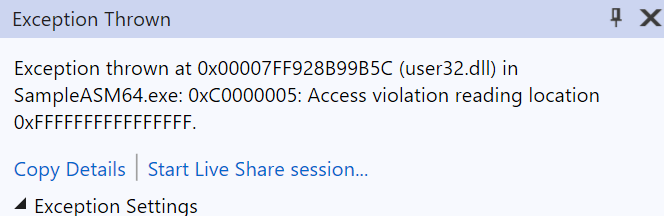
END

Здесь есть попытка изобразить структуру памяти стека вызовов на следующей диаграмме. Обратите внимание, что rsp - 28h прекрасно выровняет указатель стека по 16-байтовому выравниванию, поскольку процедура main() всегда начинается с указателя стека, адрес которого заканчивается на 8h. Вы можете убедиться в этом сами, отладив приведенный выше код, запустив его шаг за шагом и наблюдая за значениями регистров в окне регистров.



**Область стека соглашения о вызовах x64**

Чтобы убедиться в необходимости выравнивания, попробуйте изменить «sub rsp, 28h» на «sub rsp, **20h**». Это приведет к ошибке выполнения Access Violation при вызове MessageBox():



Некоторые API Win32 не такие строгие. Например, ExitProcess не вызывает исключение нарушения прав доступа. Но чтобы соблюдать соглашение о вызовах x64, каждый 64-битный ассемблерный код, вызывающий Win32 API, должен иметь какой-либо оператор, например «sub rsp, 28h», для резервирования теневой области и настройки выравнивания указателя стека.

Обратите внимание: если используется полный код пролога/эпилога x64, «sub rsp, 28h» необходимо изменить на «sub rsp, 20h», поскольку предыдущая инструкция «push» изменила rsp на 8h байтов. Значит, нужно вычесть всего 20ч.

**Обычный код пролога/эпилога x64**

.code

main PROC

; обычный код пролога для x86-64

push rbp ; сохранить вызов rbp

mov rbp, rsp ; использовать rbp в качестве указателя кадра

sub rsp, 20h ; зарезервировано как теневая область параметров Win32

; какой-то код

; обычный код эпилога для x86-64

mov rsp, rbp ; восстановить указатель стека

pop rbp ; восстановить вызов rbp

ret

main ENDP

**Задания к практической работе**

Соберите и создайте exe-файлы в VS2024 проектов из примеров проектов MASM (как для 64-битной, так и для 32-битной сборки). Их можно загрузить по ссылкам Github ниже:

<https://github.com/nthana/SampleASM64/>

<https://github.com/nthana/SampleASM32/>

В этих примерах также демонстрируется, как вызывать API-интерфейс Win32 «MessageBox» из ассемблерных кодов.