**Практическая работа № 12**

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнил | Студент гр. МФ-11 | Лазаренко А. В. | 19.05.2021 |
| Проверил | Ст. преподаватель | Меняйлов Е. С. |  |

**Тема работы:**

**«**Создание приложений на языке Ассемблера**»**

**Цель работы:**

Ознакомиться с основами создания многомодульных приложений на языке Ассемблер.

**Задание:**

1. Проанализировать арифметическое выражение, определить

порядок выполнения операций, типы получающихся значений, там,

где возможно, упростить.

2. Вычислить область определения выражения

3. Оценить значения переменных, при которых не произойдет

переполнения.

4. Составить программу на Ассемблере для вычисления выражения.

**Программа, выводящая в окно введенный с клавиатуры символ**

(Из книги Магды Ю. С. Ассемблер. Разработка и оптимизация Windows-приложений. Глава 5.6)

**Листинг файла proj\_1.asm**

. .386

.model flat, stdcall

option casemap:none ; указывает Assembler, чтобы он различал регистры, так как по умолчанию он их не различает.

; подключение библиотек

include C:\masm32\include\windows.inc

include C:\masm32\include\user32.inc

include C:\masm32\include\kernel32.inc

include C:\masm32\include\gdi32.inc

includelib C:\masm32\lib\user32.lib

includelib C:\masm32\lib\kernel32.lib

includelib C:\masm32\lib\gdi32.lib

;объявление прототипов

WinMain PROTO :DWORD, :DWORD, :DWORD, :DWORD ;это условное имя, используемое для точки входа в оконное приложение.

WndProc PROTO :DWORD, :DWORD, :DWORD, :DWORD ;функция WndProc и будет получать сообщения, которые Windows будет отсылать окну

;сегмент данных

.data

szDisplayName DB "Отображение введеного символа", 0

CommandLine DD 0

hWnd DD 0

hInstance DD 0 ;Дескриптор экземпляра, который содержит оконную процедуру для класса.

szClassName DB "Demo\_Class", 0

sTitle DB "Введенный символ", 0

sMsg DB "Вы ввели символ "

c1 DB ?, 0

;сегмент кода

.code

start:

push NULL ; помещение в стек параметров для дальнейшего вызова функции

call GetModuleHandle ; Если этот параметр равен NULL, GetModuleHandle возвращает дескриптор файла, который использовался для создания вызывающего процесса (файл .exe).

mov hInstance, EAX

call GetCommandLine ;Извлекает строку командной строки для текущего процесса.

mov CommandLine, EAX

push SW\_SHOWDEFAULT ;Устанавливает состояние показа указанного окна.

push CommandLine ; помещение в стек параметров для дальнейшего вызова функции

push NULL; помещение в стек параметров для дальнейшего вызова функции

push hInstance; помещение в стек параметров для дальнейшего вызова функции

call WinMain

push EAX

call ExitProcess ;Завершает вызывающий процесс и все его потоки.

WinMain proc hInst :DWORD, ;передаем в процедуру значения согласно прототипу

hPrevInst :DWORD,

CmdLine :DWORD,

CmdShow :DWORD

LOCAL wc :WNDCLASSEX ; создаём локальный экземпляр, для обращения к членам класса WNDCLASSEX

LOCAL msg :MSG ;создаём локальный экземпляр, для обращения к членам класса MSG

mov wc.cbSize, sizeof WNDCLASSEX ; Устанавливает размер этой структуры, в байтах.

mov wc.style, CS\_HREDRAW or CS\_VREDRAW ; перерисовка окна при изменении его размеров

mov wc.lpfnWndProc, offset WndProc ; Указатель на оконную процедуру

mov wc.cbClsExtra, NULL ;Устанавливает число дополнительных байт, которые размещаются вслед за структурой класса окна.

mov wc.cbWndExtra, NULL ;Устанавливает число дополнительных байтов, которые размещаются вслед за экземпляром окна.

push hInst ; помещаем в стек дескриптор окна

pop wc.hInstance ; hInstance - Дескриптор экземпляра, который содержит оконную процедуру для класса.

mov wc.hbrBackground, COLOR\_BTNFACE+9 ; hbrBackground - Дескриптор кисти фона класса. Если дается код цвета, Вы должны преобразовать его в один из ниже перечисленных типов HBRUSH

mov wc.lpszMenuName, NULL ; lpszMenuName - Указатель на символьную строку с символом конца строки (нуль-терминатором), которая устанавливает имя ресурса меню класса, которое как имя показывается в файле ресурса

mov wc.lpszClassName, offset szClassName ; lpszClassName - Указатель на символьную строку с нулем в конце

invoke LoadIcon, NULL, IDI\_APPLICATION ; помещаем иконку по умолчанию(IDI\_APPLICATION) в стек и вызваем функцию LoadIcon

mov wc.hIcon, EAX ;hIcon - Дескриптор значка класса.

invoke LoadCursor, NULL, IDC\_ARROW; помещаем курсор по умолчанию(IDC\_ARROW) в стек и вызваем функцию LoadIcon

mov wc.hCursor, EAX ; Дескриптор курсора класса.

mov wc.hIconSm, 0 ;Дескриптор маленького значка, который связан с классом окна.

invoke RegisterClassEx, ADDR wc ; проверка инициализации каждого поля(RegisterClassEx) переменной wc

; Функция CreateWindowEx создает перекрывающее, выпрыгивающее или дочернее окно с расширенным стилем

invoke CreateWindowEx, WS\_EX\_OVERLAPPEDWINDOW, ADDR szClassName, ADDR szDisplayName, WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT,

CW\_USEDEFAULT, NULL, NULL, hInst, NULL

;WS\_EX\_OVERLAPPEDWINDOW - Комбинация стилей рамки с выпуклым и притопленным краем.

;szClassName - указатель на зарегистрированное имя класса

;szDisplayName - название выводимого окна

;CW\_USEDEFAULT - параметры окна заданные по умолчанию

;hInst - Дескриптор экземпляра модуля, который будет связан с окном.

mov hWnd, EAX ; Дескриптор окна, оконная процедура которого принимает сообщение.

invoke ShowWindow, hWnd, SW\_SHOWNORMAL; вызыввем оконо по умолчанию

invoke UpdateWindow, hWnd; обновляем окно при его изменении

StartLoop:

push 0

push 0

push NULL

lea EAX, msg

push EAX

call GetMessage; помещаем в структуру msg сообщение из вызывающего потока

cmp EAX, 0

je ExitLoop

lea EAX, msg

push EAX

call TranslateMessage; переводит нажатие клавиши в ее символьное представление

lea EAX, msg

push EAX

call DispatchMessage; Отправляет сообщение оконной процедуре.

jmp StartLoop

ExitLoop:

mov EAX, msg.wParam ; Определяет дополнительную информацию о сообщении.

ret

WinMain endp; заканчиваем процедуру обработки основного окна

; начинаем работу с окном вывода символа

WndProc proc hWin :DWORD,

uMsg :DWORD,

wParam :DWORD,

lParam :DWORD

LOCAL hdc :HDC ; создаем переменную типа структуры HDC

LOCAL ps :PAINTSTRUCT ; создаем переменную типа структуры PAINTSTRUCT

cmp uMsg, WM\_PAINT ; сравниваем распоряжение на перерисовку окна

jne next\_1

invoke BeginPaint, hWnd, ADDR ps ; Функция BeginPaint готовит заданное окно к окрашиванию и заполняет структуру ps информацией об окрашивании.

mov hdc, EAX

lea EDX, ps

push EDX

push hWnd

call EndPaint ; Функция EndPaint отмечает конец окрашивания в заданном окне.

ret

next\_1:

cmp uMsg, WM\_CHAR

jne next\_2

mov EAX, wParam

mov c1, AL

push MB\_OK

push offset sTitle

push offset sMsg

push 0

call MessageBox ; Функция MessageBox создает, показывает на экране и использует окно сообщения.

ret

next\_2:

cmp uMsg, WM\_DESTROY ; Сообщение WM\_DESTROY отправляется тогда, когда окно закрываем

jne next\_3

push NULL

call PostQuitMessage ; Функция PostQuitMessage указывает системе, что поток сделал запрос на то, чтобы завершить свою работу

xor EAX, EAX

ret

next\_3:

push lParam

push wParam

push uMsg

push hWin

call DefWindowProc; Функция DefWindowProc вызывается оконной процедурой по умолчанию, чтобы обеспечить обработку по умолчанию любого сообщения окна, которые приложение не обрабатывает.

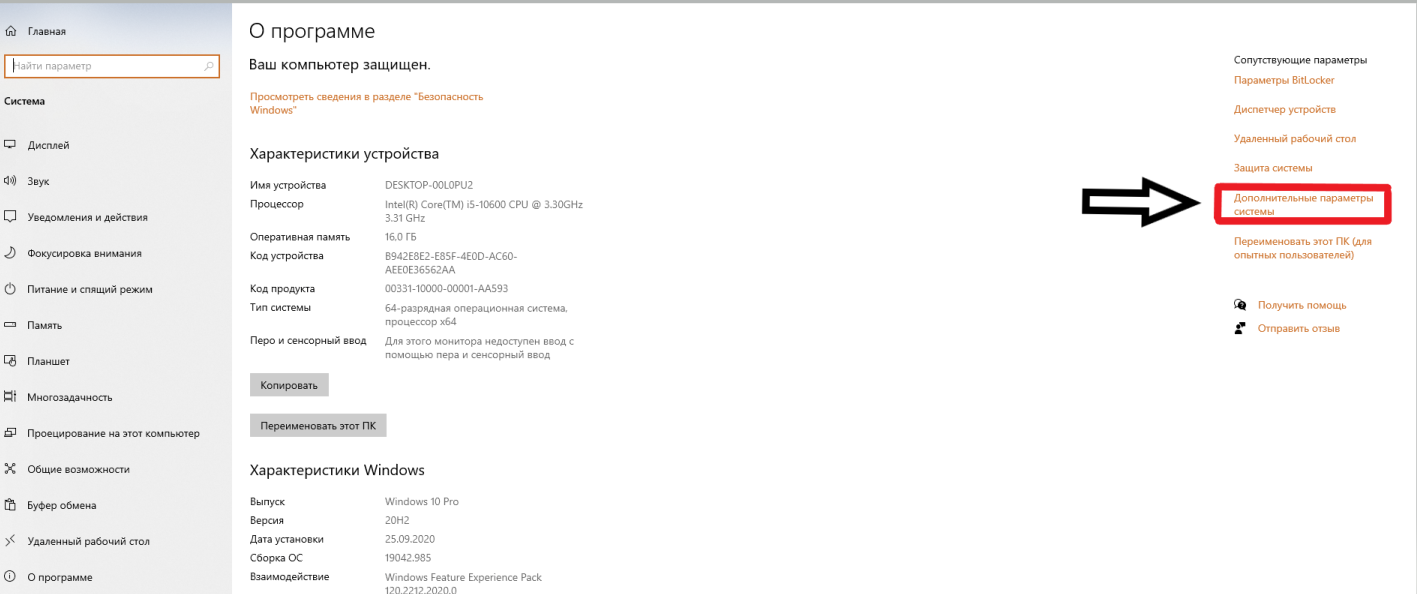
ret

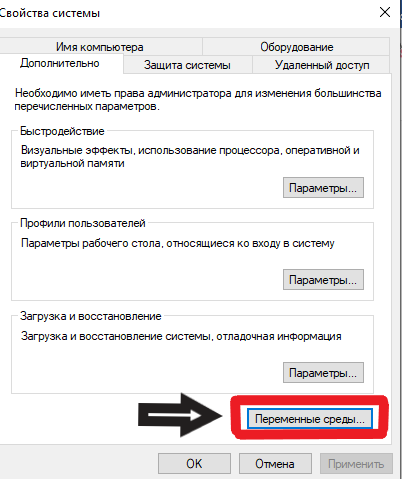
WndProc endp ; заканчиваем процедуру обработки дочернего окна

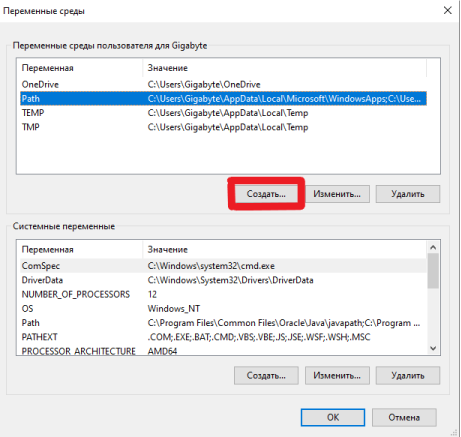
end start ;точка выхода из программы

**Важное замечание**

Перед процессом компиляции, требуется добавить путь в папку «**bin**» каталога MASM32 в системные переменные (так как именно там и находятся нужные для компиляции **ml.exe** и **link.exe**). Сделать это можно, перейдя в пункт «Панель управления» -> «Система» -> «Дополнительные параметры системы» -> «Переменные среды» -> «Path» -> «Создать», вставив нужный путь (в моем случае, **C:\masm32\bin**).







**Компиляция**

Для компиляции я использую возможности MASM32, а именно:

1) **ml –** транслятор **asm**-файлов

**ml** не создаёт готовую программу под конкретную операционную систему, и её конкретный формат. Он делает промежуточный объектный файл с расширением «.**obj**».

Ключ-параметр **/c** необходим, чтобы происходила ТОЛЬКО сборка без компоновки (**asm**-файла).

Ключ-параметр **/coff** (сокращение от **Common Object File Format**)– указывает на формат исполняемого файла (**exe**).

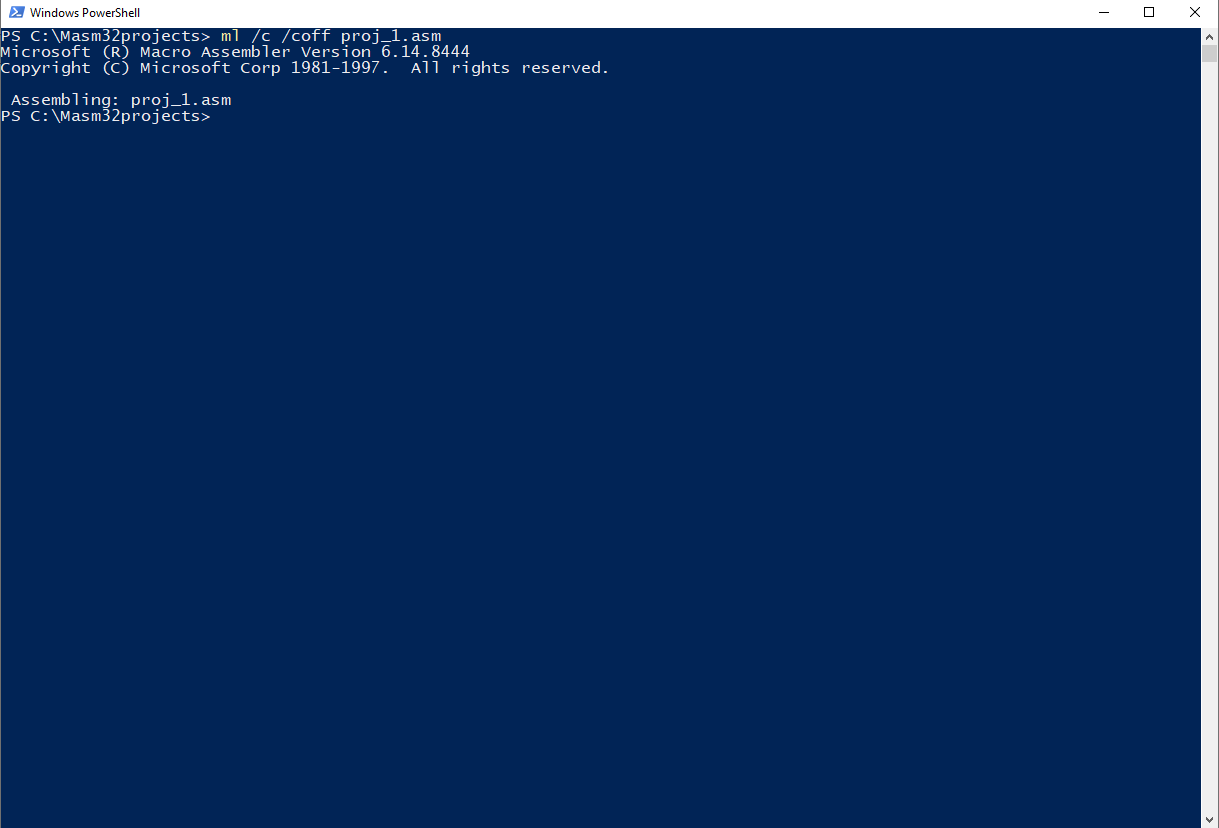
2) **link –** компоновщик **obj**-файлов

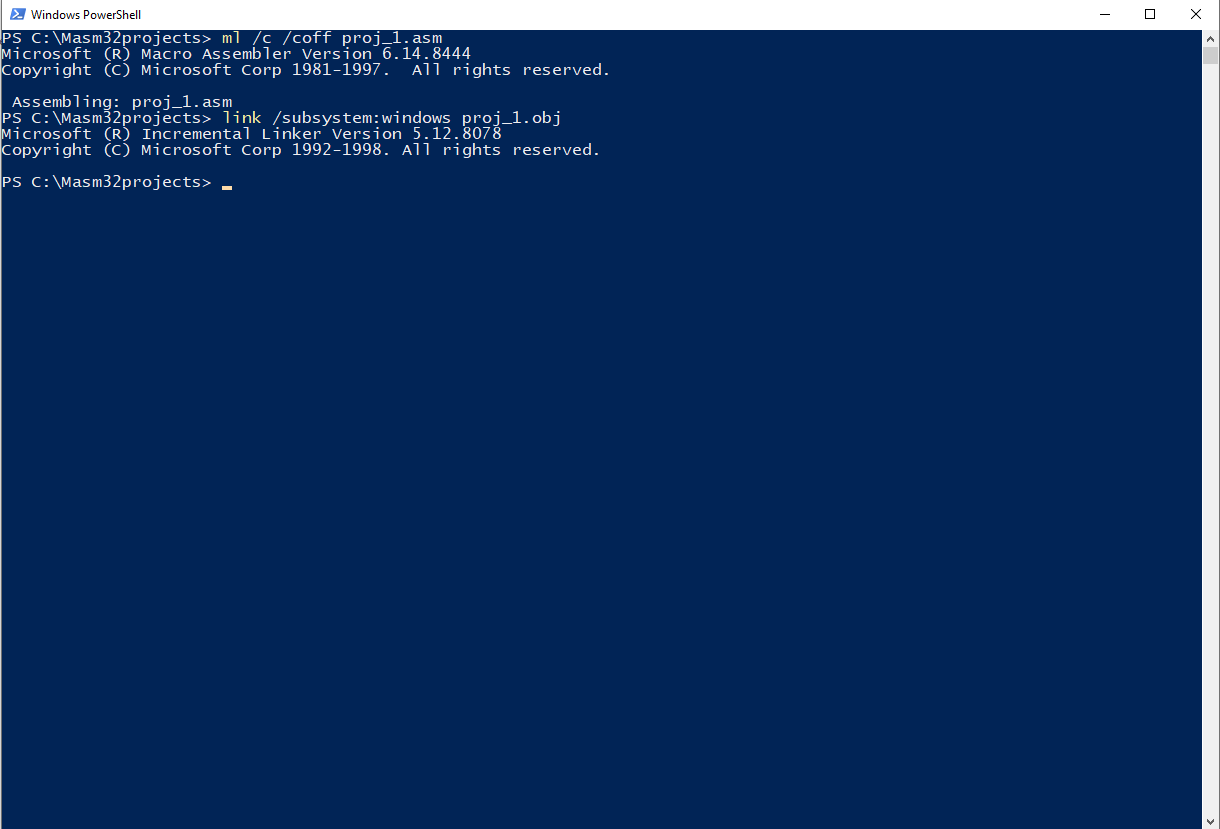
**link** необходим для создания исполняемых файлов в формате «**exe**».

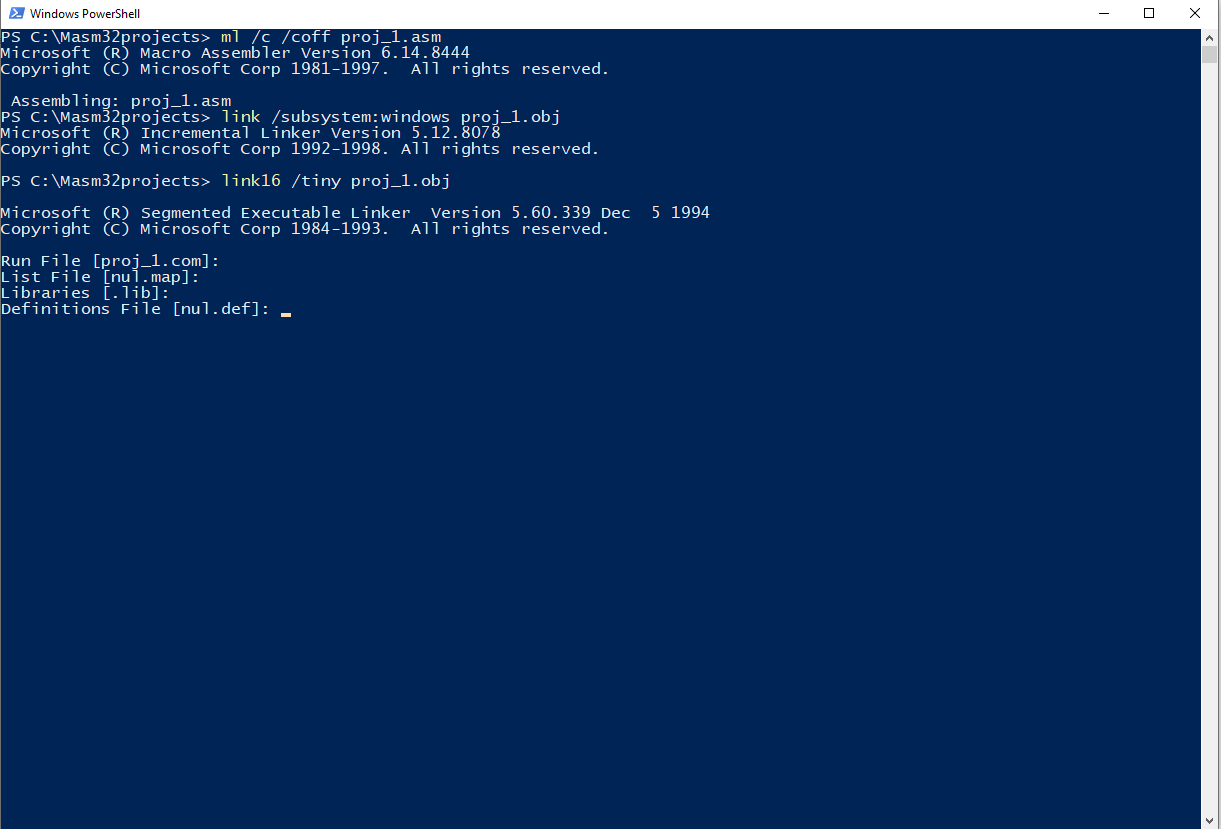
Ключ-параметр **/subsystem:windows** необходим, чтобы дать явное указание операционной системе на то, что данное приложение не консольное, а оконное.

3) **link16 -** 16-битный компоновщик **obj**-файлов

**link16** необходим для создания DOS-программ в формате **com**.

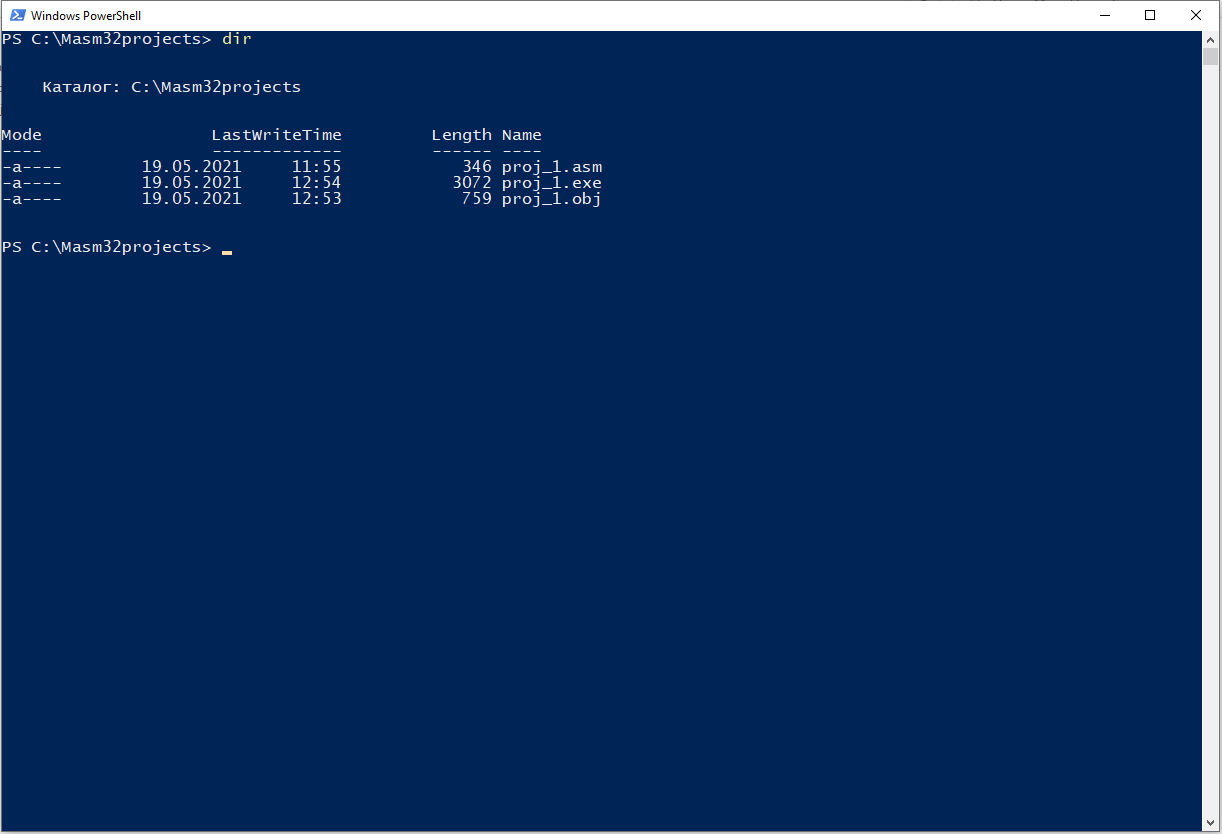




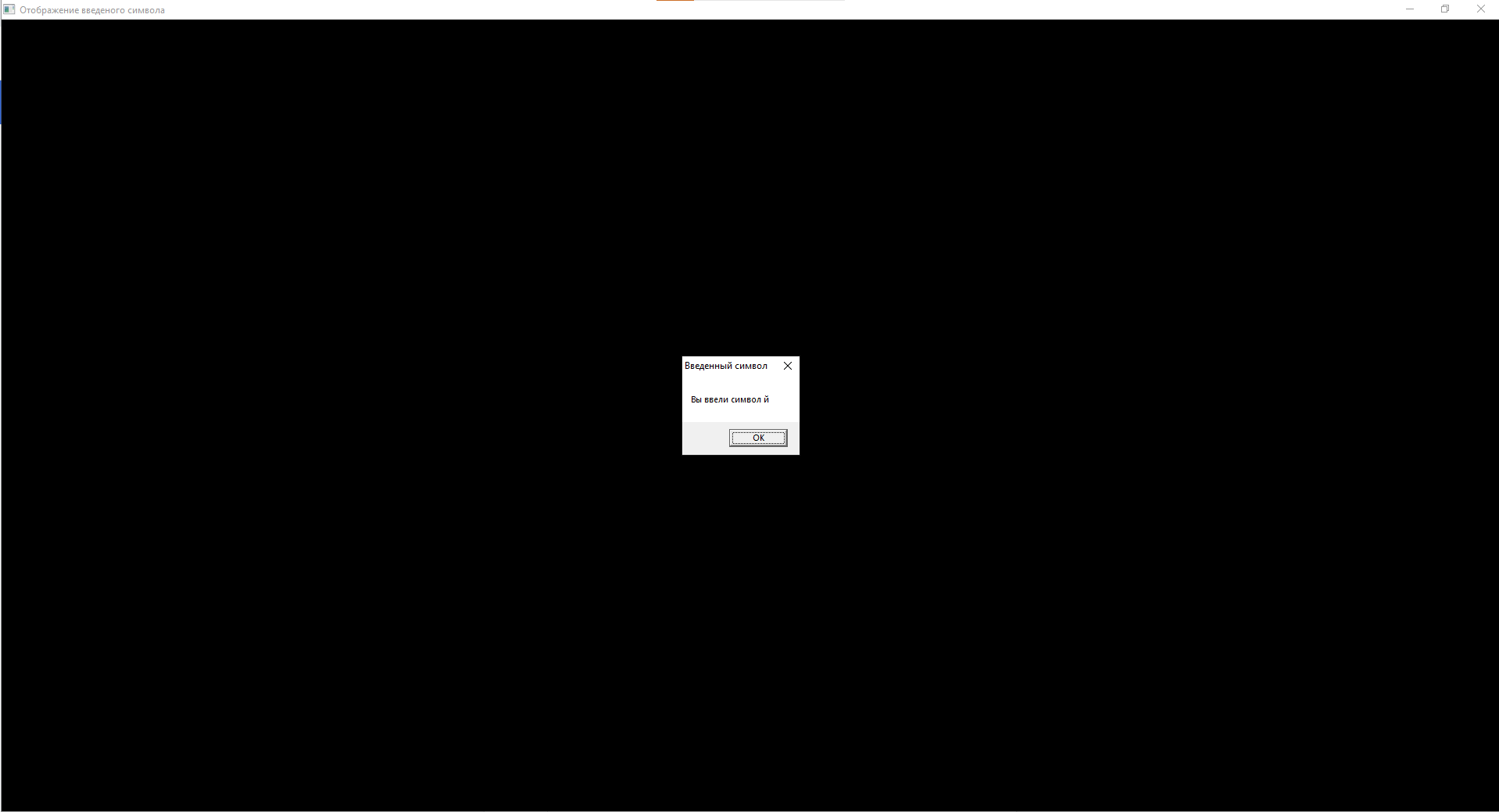


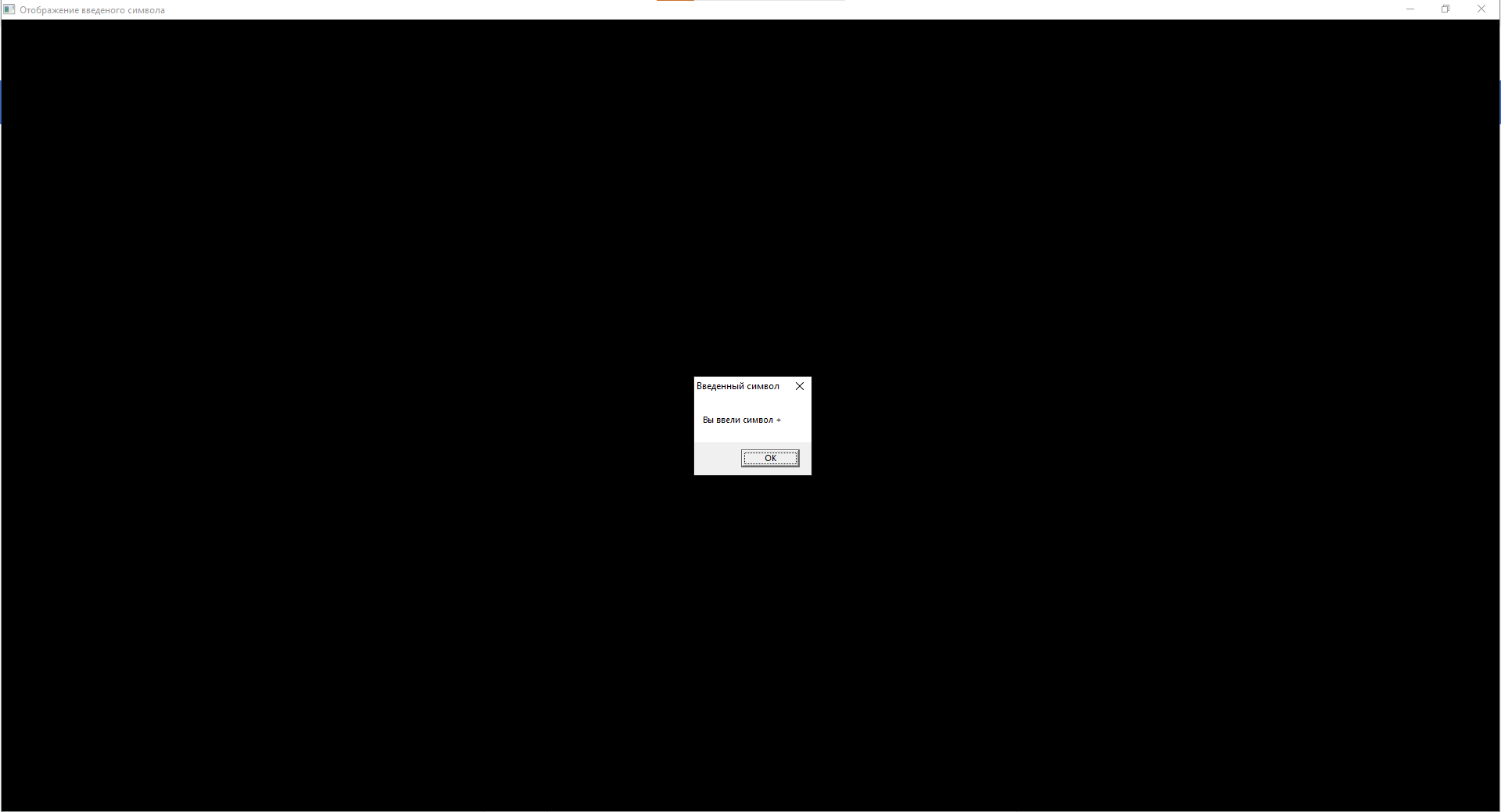
**Проверка работоспособности**

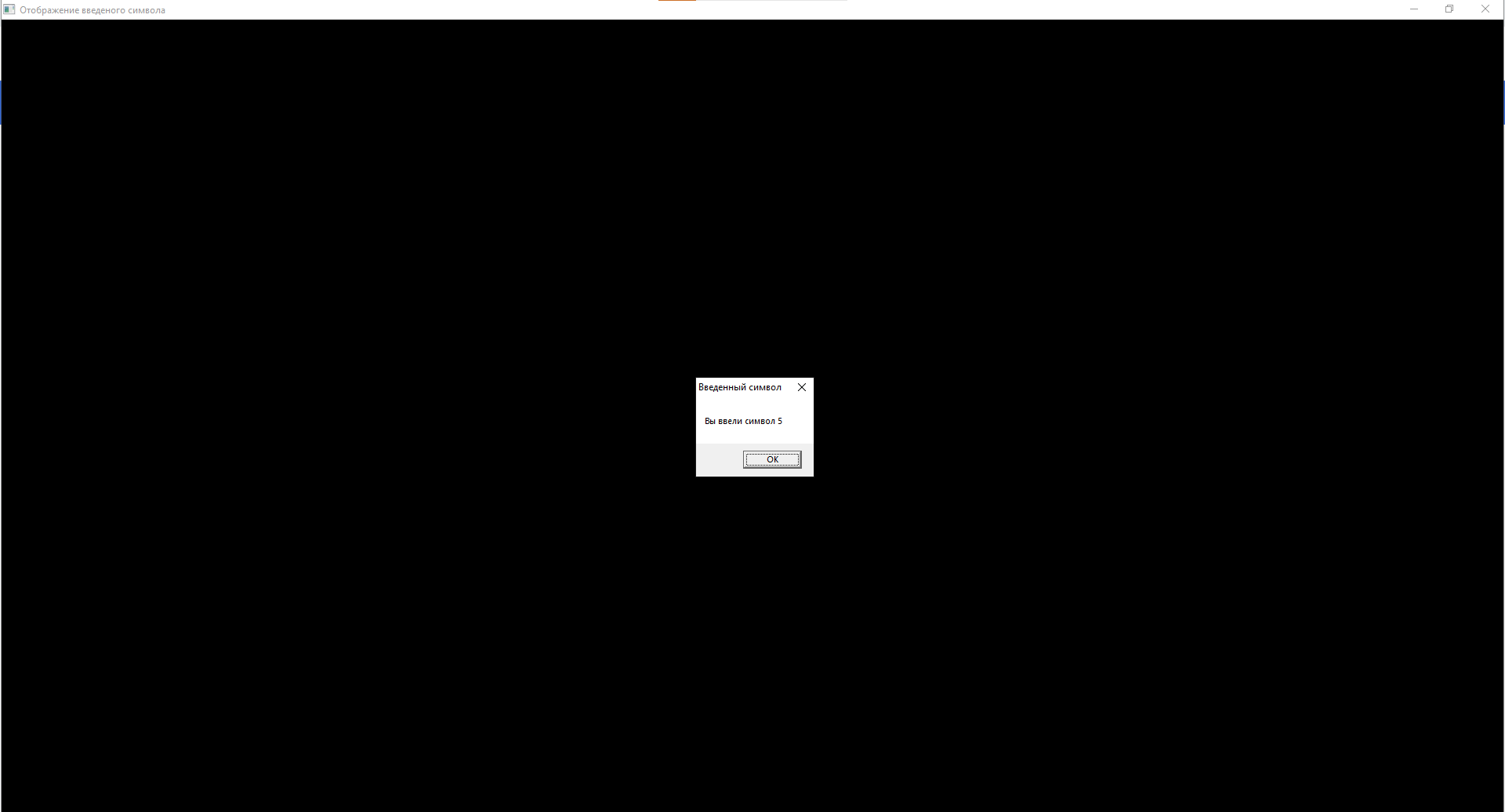
Перейдем на диск C, в папку Masm32projects, и проверим наличие необходимых файлов, в частности, исполняемого **exe**-файла.



Все в порядке, теперь запускаем **exe**-файл:







Очевидно, исполняемый файл выполнил именно то, что и задумывалось – вывод введенного символа с клавиатуры.

**Вывод**

В результате практической работы **№12** я реализовал вывод символа, введенного с клавиатуры (в кодировке ANSI), в окне, а также с помощью средств MASM32 научился создавать и обрабатывать **asm**-, **obj**- и **com**-файлы. В конце работы, я сумел проверить работоспособность исполняемого файла программы, и убедиться в правильности выполненных действий.