Практика 2: изучение ввода/вывода информации в консоли.

Для начала укажем нужный формат исполняемого файла. Первой строкой пишем

format PE Console

Компилируем и запускаем. Теперь открывается пустое окно консоли. Займемся выводом информации в консоль. Для начала заменим вывод строки в всплывающее окно выводом на консоль. В наших практиках будем использовать только системные вызовы Win32API. Самым полным справочником по документированным вызовам Win32API следует признать раздел сайта самой MicroSoft - https://docs.microsoft.com/en-us/windows/console/console-functions . Разумеется использовать можно любой доступный справочник. Для вывода в консоль существует несколько системных вызовов

- WriteConsole
- WriteConsoleOutput
- WriteConsoleOutputAttribute
- WriteConsoleOutputCharacter

Использовать будем самый простой - **WriteConsole**. Он имеет 5 параметров

- дескриптор экранного буфера — дескриптор нашего консольного приложения
- буфер записи — собственно строка, выводимая на экран
- число символов записи — сколько символов выводить на экран
- число фактически записанных символов — переменная, в которую системный вызов запишет сколько символов он фактически вывел на консоль
- зарезервированное значение — не влияет на работу

команды

Таким образом, для вывода на экран строки нужна переменная со строкой и дескриптор потока вывода текущего окна. Дескриптор потока вывода получаем вызовом **GetStdHandle** с параметром STD_OUTPUT_HANDLE. В целом код вывода строки на консоль получается такой

```
.data
hellostr DB “Hello world.”,0
inputnumber DW 0

.code

push STD_OUTPUT_HANDLE
call [GetStdHandle]
push 0
push inputnumber
push 12
push hellostr
push eax
call [WriteConsole]
```

Вопрос вызывает только строка ***push eax*** . В регистре eax хранилось значение, которое вернул системный вызов GetStdHandle.

Для ввода строки у нас есть тоже несколько системных вызовов

- ReadConsole
- ReadConsoleInput
- ReadConsoleInputEx

Отличающиеся параметрами вызова и – соответственно – немного отличающиеся результатами применения.

Воспользуемся системным вызовом **ReadConsole** со следующими

параметрами

- дескриптор буфера ввода консоли
- буфер данных — строка для размещения вводимых данных
- число символов для чтения — сколько планируется получить символов со ввода
- число прочитанных символов — сколько фактически введено символов
- зарезервированное значение

Дескриптор потока ввода получается тем же вызовом GetStdHandle, только с параметром STD_INPUT_HANDLE. В итоге код выглядит так

```
.code
inputstr DB "  ",0

.code
push STD_INPUT_HANDLE
call [GetStdHandle]
push 0
push inputnumber
push 8
push inputstr
push eax
call [ReadConsole]
```

Заметим, что строка вводится сразу за выведенной ранее строкой. Дело тут в том, что необходимо после вывода строки переводит каретку на новую строку и возвращать её в начало строки. В консоли для этого служат два спец символа с кодами 10 и 13. Достаточно их добавить в конец выводимой строки.

```
.data
hellostr DB "Hello world.",10,13,0
```

и не забыть их включить в длину выводимого сообщения

push 14

вместо 12. Более подробно с применением спец символов можно ознакомится в документации по magic-bytes консоли. Нам другие символы не потребуются.
Так же мы не рассматриваем ещё три системных вызова

- ReadConsoleOutput
- ReadConsoleOutputAttribute
- ReadConsoleOutputCharacter

Они позволяют получать информацию о том, что сейчас выведено на экране консоли. Например, ReadConsoleOutputCharacter позволяет получить набор символов, начинающихся с определенных координат экрана. Способ применения подобных вызовов предлагается додумать самостоятельно.

Задание1:
запросить имя пользователя и вывести его на экран со следующей строки.
Задание 2:
вывести в конце программы запрос на нажатие enter

Цвет текста в консоли можно задавать функцией
SetConsoleTextAttribute , имеющей два параметра

- дескриптор экранного буфера консоли
- атрибут символа

Заданные атрибуты буду применяться ко всем новым символам на консоли, не изменяя атрибуты уже выведенных символов. Атрибут символа можно задавать при помощи команд, но проще при помощи 16битного значения. Младшие 4 бита отвечают за цвет символа

- 0 — чёрный

- 1 — синий
- 2 — зеленый
- 3 — синезеленый
- 4 — красный
- ….
- F - белый

Следующие 4 бита отвечают за цвет фона символа. Таблица цветов сохраняется.
Значит, чтобы вывести надпись красным цветом достаточно перед её выводом вызвать следующий код

```
push STD_OUTPUT_HANDLE
call [GetStdHandle]
push 0x0004
push eax
call [SetConsoleTextAttribute]
```

Макросы

Для многократного вызова одного и того же кода можно воспользоваться макросами. Макросы объявляются вне сегментов программы, например перед сегментом кода.
Объявление происходит ключевым словом macro, именем макроса и – опционально – параметрами макроса, а обращение простым вызовом имени макроса. Например, смена цвета символа будет выглядеть так

```
macro setcolor color
{
push STD_OUTPUT_HANDLE
call [GetStdHandle]
push color
```

push eax
call [SetConsoletextAttribute]
}

Здесь считаем что код цвета располагается в регистре dx. Вызов макроса будет выглядеть так

setcolor <код цвета>

Задание 3:
вывести на экран надпись COLOR всеми 15 возможными цветами.
Решение:
Реализуем два макроса: установку цвета и вывод строки на экран и вызываем из 16 раз

macro writecolor
{

}
macro setcolor color
{
....
writecolor
}

Для дальнейшей работы нам понадобится цикл. Подробнее с организациями циклов мы ознакомимся позднее. Используем пока следующий вариант цикла:

метка_начала_цикла:

cmp регистр,константа
jz метка_конца_цикла
....
jmp метка_начала_цикла
метка_конца_цикла:

где cmp выполняет сравнение регистра с константой, то есть фактически проверяет условие. JZ переходит на метку, если условие выполнилось. JMP выполняет безусловный переход на начала цикла после выполнения тела цикла.
Вариант решения через цикл следующий:

```
xor ecx,ecx
showcolors:
cmp ecx,16
jz endcolors
push ecx

setcolor ecx

push STD_OUTPUT_HANDLE
call [GetStdHandle]
push 0
push inputnumber
push 7
push colorstr
push eax
call [WriteConsole]

pop ecx
inc ecx
jmp showcolors
endcolors:
```

где inc обозначает очевидно понятную функцию инкремента на 1. В .data не забываем строку определить

colorstr DB “Color”,10,13,0

Дополнительно может пригодится системный вызов установки заголовка окна

Push titlestr
Call [SetConsoleTitle]

Последней коснемся темы режима консольного ввода. Два системных вызова

1. GetConsoleMode – получить режим консоли
2. SetConsoleMode – установить режим консоли

У обоих параметрами выступают

- hConsoleHandle – дескриптор ввода или вывода
- dwMode – режим консоли: комбинация флагов, задающая режим поведения ввода или вывода консоли

Биты dwMode задают режимы работы консоли:

- отображение вводимых символов
- копирование или перемещение вводимых символов из буфера ввода
- режим работы ReadConsole и WriteConsole
- режим вывода информации
- и прочее.

Для нас пока необходимо, что режим равный 0 будет задавать вводи информации без отображения на экране и без подтверждения ввода символом Enter – как раз то, что подходит для задержки программы до нажатия любой клавиши. Переключим консоль в соответствующий режим

```
        push STD_INPUT_HANDLE
        call [GetStdHandle]
        push NULL
        push eax
        call [SetConsoleMode]
```

Задание 4:
Вывести в конце программы Press any key to exit… и выйти из программы по нажатию 1 любой клавиши.
Решение:
лучше реализовать отдельным макросом, который сначала выводит строку, потом запрашивает ввод

```
.data
        exittext db 'Press any key to exit...',0
        inputnum dw 0
macro exitprogram
{
        push STD_OUTPUT_HANDLE
        call [GetStdHandle]
        push 0
        push inputnumber
        push 12
        push exittext
        push eax
        call [WriteConsole]
        push STD_INPUT_HANDLE
        call [GetStdHandle]
        push NULL
        push eax
        call [SetConsoleMode]
        push STD_INPUT_HANDLE
        call [GetStdHandle]
```

```
        push 0
        push inputnum
        push 1
        push exittext
        push eax
        call [ReadConsole]
        push 0
        call [ExitProcess]
}
.code
start :

....

        exitprogram
.end start
```

Домашнее задание:

Реализовать вывод цветных строк на экран с паузой в 1 секунду. Для реализации паузы воспользоваться системным вызовом sleep с параметром время ожидания в миллисекундах.