Programming in Assembler

Laboratory manual

Exercise 7

MASM 8.0 – programming assembler in Visual Studio 2005

Exercise goal:

Students get familiarized with Microsoft Macro Assembler 8.0 (MASM) Package (x86) and Visual Studio 2005 Express Edition – a free, integrated development environment from Microsoft – by building sample Windows console application.

1. MASM 8.0 features.

The latest MASM (8.0) brings possibilities on programming the latest features of modern processors:

- MASM expressions are now 64-bit values. In previous versions MASM expressions were 32-bit values.
- The instruction __asm int 3 now causes a function to be compiled to native. For more information, see __asm.
- ALIAS (MASM) is now documented.
- /ERRORREPORT ml.exe and ml64.exe option is added.
- .FPO is now documented.
- H2INC.exe will not ship in Visual C++ 2005. If you need to continue to use H2INC, use H2INC.exe from a previous version of Visual C++.
- operator `IMAGEREL` has been added.
- operator `HIGH32` has been added.
- operator `LOW32` has been added.
- ml64.exe is a version of MASM for the x64 architecture. It assembles x64 .asm files into x64 object files. Inline assembly language is not supported in the x64 compiler. For more information, see MASM for x64 (ml64.exe).
- The following MASM directives have been added for ml64.exe (x64):
- `.ALLOCSTACK`
- `.ENDPROLOG`
- `.PUSHFRAME`
- `.PUSHREG`
- `.SAVEREG`
- `.SAVEXMM128`
- `.SETFRAME`
- In addition, the PROC directive was updated with x64-only syntax.
- MMWORD directive has been added
- /omf (ML.exe command line option) now implies /c. ML.exe does not support linking OMF format objects.
- The `SEGMENT` directive now supports additional attributes.
- operator `SECTIONREL` has been added.
- `XMMWORD` directive has been added

2. Installing & using MASM:

To provide successful MASM 8.0 one should install Visual C++ 2005 Express Edition (English version) first.
Both MASM and Visual Studio C++ Express Edition can be downloaded from Microsoft download center (www.microsoft.com/download).

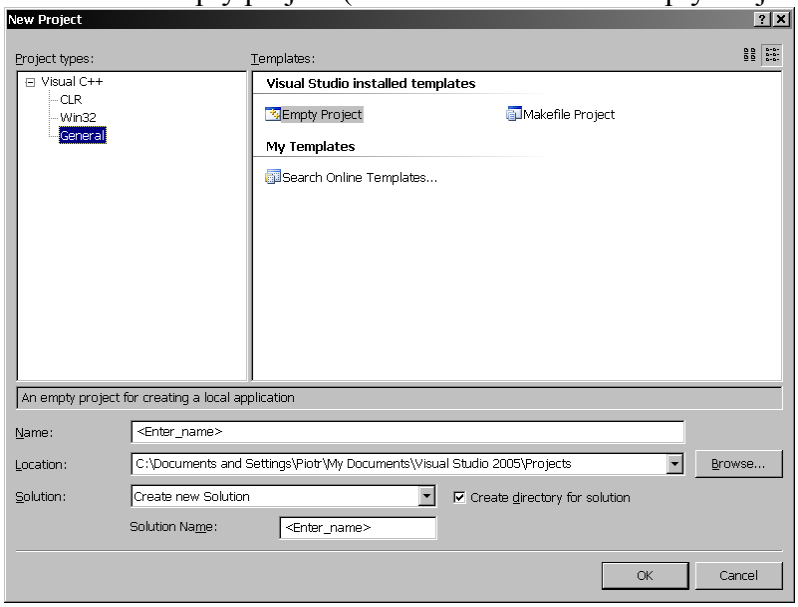
1. Installing VC++ 2005 EE takes few steps:
   - downloading and executing vcsetup.exe installer (about 2,9MB),
   - downloading (automated by running vcsetup.exe) VC++ EE core binaries, thus active Internet connection is necessary during setup,
   - downloading and running VC++ 2005 EE SP1 and upgrading installation.

2. Installing MASM 8.0 assembler binaries to the VC++'s \bin directory (automated by running MASMsetup.EXE (about 0.3MB).

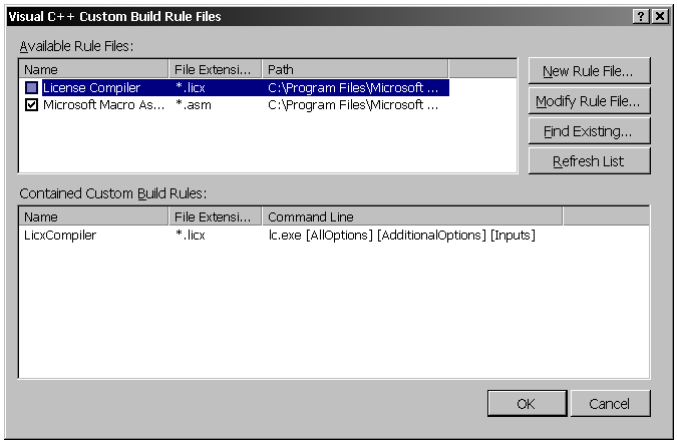
3. Creating sample program from scratch

a. Create empty solution

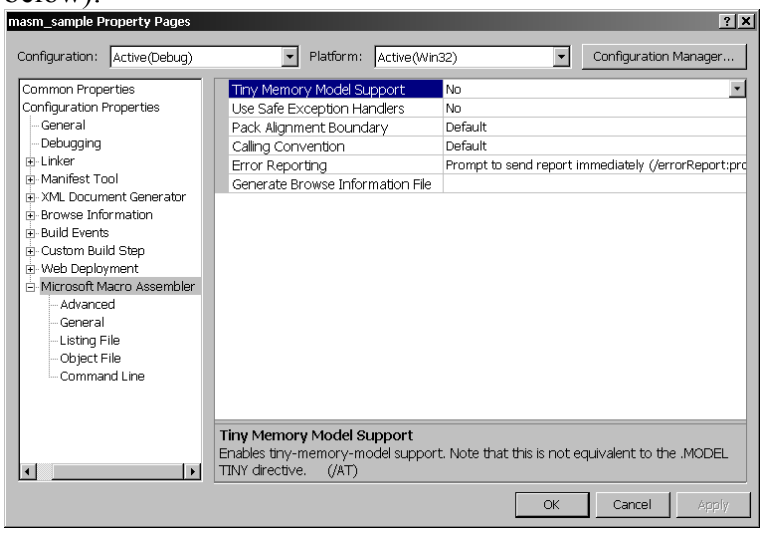
Create new empty project (Visual C++/General/Empty Project):



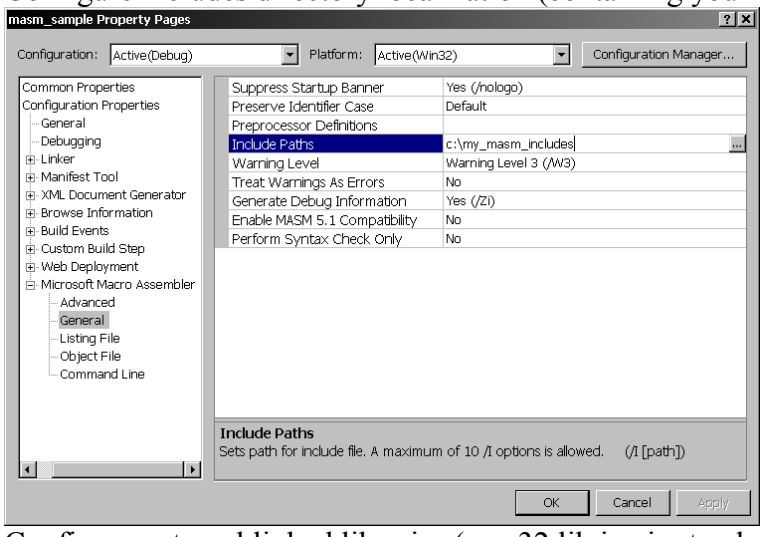
b. Add MASM support:
Select **Custom Build Rules** in PROJECT menu and check **Microsoft Macro Assembler**:



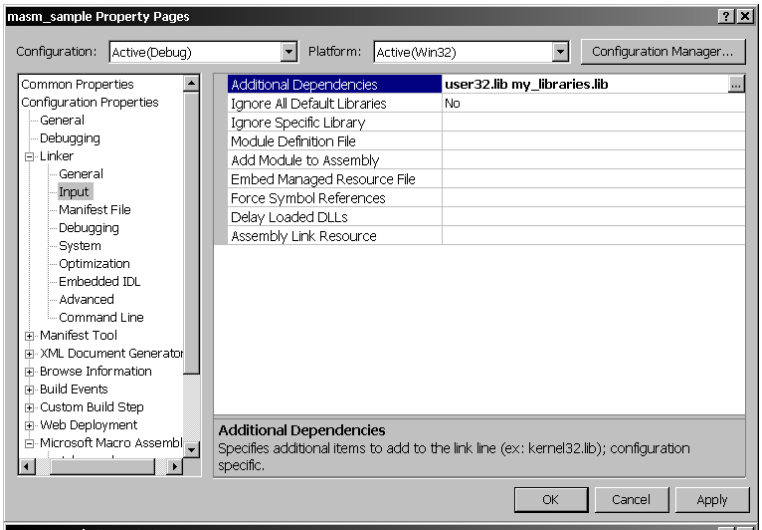
c. **Configure project properties**:
Right-click the project title (entered in step a.) within "Solution Explorer" browser tree, select "Properties", observe "Microsoft Macro Assembler" position within left tree panel (similar to picture below):

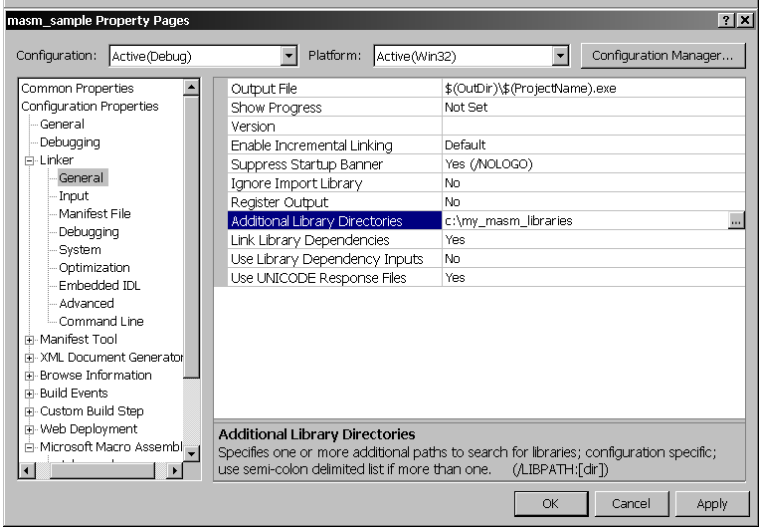


Configure includes directory localization (containing your *.inc files, if any):

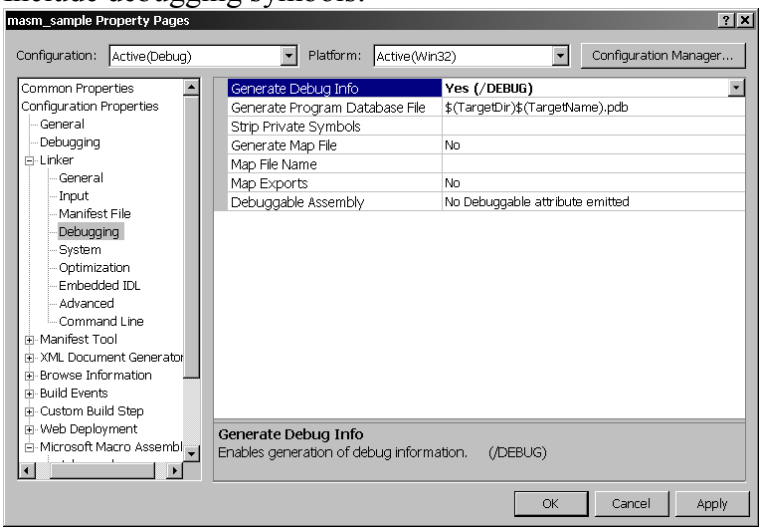


Configure external linked libraries (user32.lib i.e. is standard Windows file however is not necessary for this sample). Attach other libraries (*.lib) if necessary in your final linked executable, separate names with space. Also provide path to the folder containing libraries:

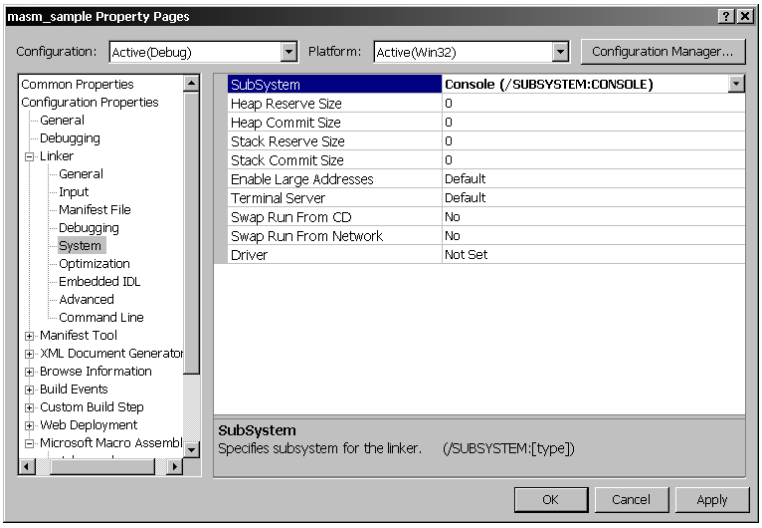




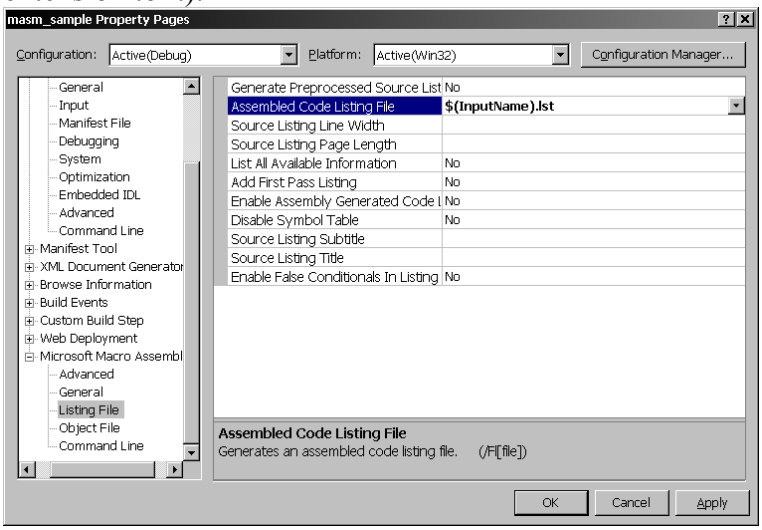
Include debugging symbols:



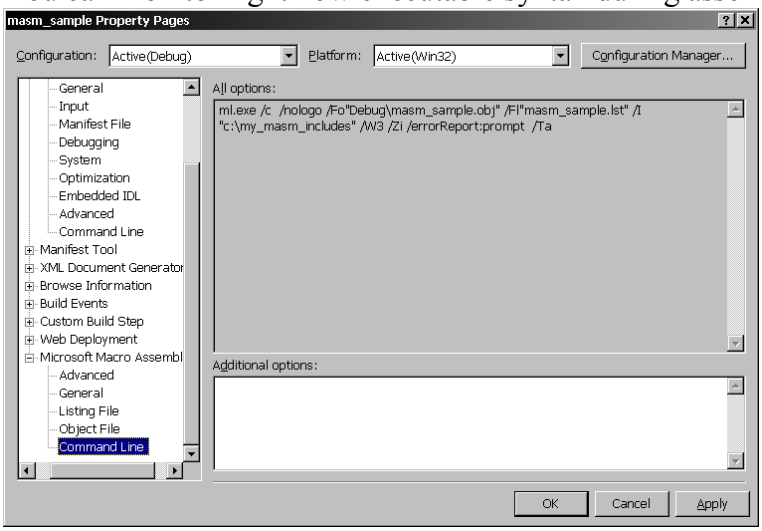
Define system for linker output:



Include listing (.lst) file (write as below or chose <Edit> and select appropriate Macro + add .lst extension to it):



You can monitor right now executable syntax during assembling by choosing:



d. Customize environment

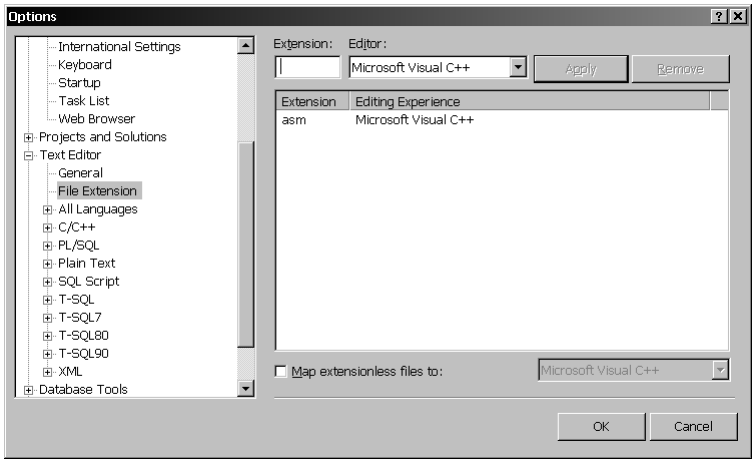
Force VS IDE to highlight MASM syntax.

To bring it to the life you need to create *usertype.dat* file in the “C:\Program Files\Microsoft Visual

Studio 8\Common7\IDE\" directory (when installed in default location). The file shall contain a list of keywords to be highlighted, one per row, i.e.:

```
…
eax
ebp
ebx
ecx
edi
edx
es
esi
esp
fs
…
```

Then teach VS IDE to recognize .ASM files as source code. Choose Tools/Options and add ASM extension as C++ source file:



Restart IDE afterwards.
Sample *usertype.dat* file is available on local server (with those document) as well as can be downloaded from Web.

e. Create source file

Create empty source file with .ASM extension and attach it to the project (right-click "Source Files", choose "Add" and then "Existing Item").

f. Edit source file

Double click source file. When opened edit the contents. Write sample program (code below):

```
TITLE Hello World in Win32 ASM (helloworld.asm)
.386
.MODEL flat, stdcall
.STACK 4096
; ------------------------------------------------------------------------
; These are prototypes for functions that we use
; from the Microsoft library Kernel32.lib.
; ------------------------------------------------------------------------
; Win32 Console handle
STD_OUTPUT_HANDLE EQU -11 ; predefined Win API constant (magic)
STD_INPUT_HANDLE equ -10 ;
GetStdHandle PROTO, ; get standard handle
nStdHandle:DWORD ; type of console handle
```

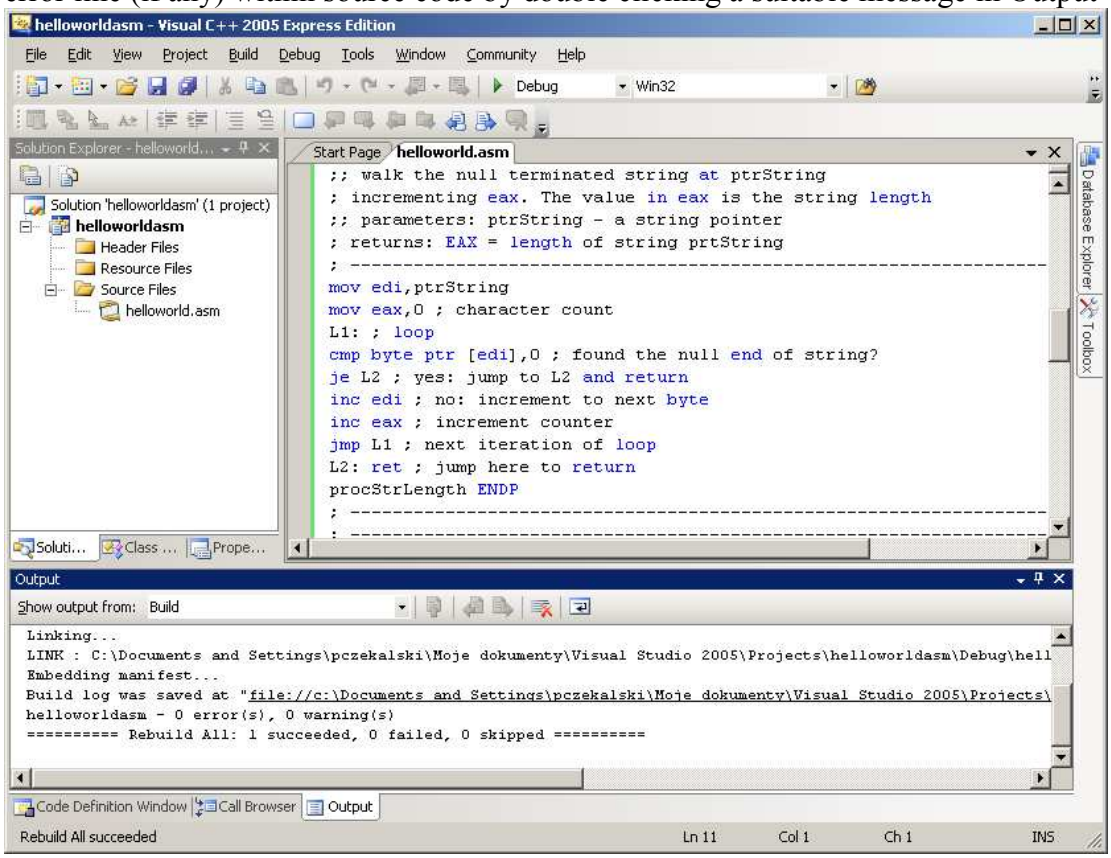
```
WriteConsole EQU <WriteConsoleA> ;
WriteConsole PROTO, ; write a buffer to the console
      handle:DWORD, ; output handle
      lpBuffer:PTR BYTE, ; pointer to buffer
      nNumberOfBytesToWrite:DWORD, ; size of buffer
      lpNumberOfBytesWritten:PTR DWORD, ; num bytes written
      lpReserved:DWORD ; (not used)

ReadConsole EQU <ReadConsoleA> ;
ReadConsole PROTO, ; write a buffer to the console
      handle:DWORD, ; output handle
      lpBuffer:PTR BYTE, ; pointer to buffer
      nNumberOfBytesToWrite:DWORD, ; size of buffer
      lpNumberOfBytesWritten:PTR DWORD, ; num bytes written
      lpReserved:DWORD ; (not used)

SetConsoleTitle EQU <SetConsoleTitleA>
SetConsoleTitle PROTO, ;
      pString:PTR BYTE        ; points to string

ExitProcess PROTO, ; exit program
      dwExitCode:DWORD ; return code
; -----------------------------------------------------------------------------
; -----------------------------------------------------------------------------
; global data
; -----------------------------------------------------------------------------
.data
consoleOutHandle dd ? ; DWORD: handle to standard output device
consoleInHandle dd ? ;
bytesWritten dd ? ; DWORD: number of bytes written
bytesRead dd ?;
buffer db 4 dup(?);
message db "Hello World",13,10,0 ; BYTE: string, with \r, \n, \0 at the end
consoleTitle db "Program testowy",0;
; -----------------------------------------------------------------------------
.code
; -----------------------------------------------------------------------------
procStrLength PROC USES edi,
ptrString:PTR BYTE ; pointer to string
;; walk the null terminated string at ptrString
; incrementing eax. The value in eax is the string length
;; parameters: ptrString - a string pointer
; returns: EAX = length of string prtString
; -----------------------------------------------------------------------------
mov edi,ptrString
mov eax,0 ; character count
L1: ; loop
cmp byte ptr [edi],0 ; found the null end of string?
je L2 ; yes: jump to L2 and return
inc edi ; no: increment to next byte
inc eax ; increment counter
jmp L1 ; next iteration of loop
L2: ret ; jump here to return
procStrLength ENDP
; -----------------------------------------------------------------------------
; -----------------------------------------------------------------------------
procWriteString proc
;;Writes a null-terminated string pointed to by EDX to standard
; output using windows calls.
; -----------------------------------------------------------------------------
```

```
pushad
INVOKE procStrLength,edx ; return length of string in EAX
cld ; clear the direction flag
; must do this before WriteConsole
INVOKE WriteConsole, consoleOutHandle, edx, eax, offset bytesWritten, 0
popad
ret
procWriteString endp
; -----------------------------------------------------------------------------
; -----------------------------------------------------------------------------
procReadKey proc
;; Reads a key from stdin
pushad
cld;
mov eax, 1;
INVOKE ReadConsole, consoleInHandle, edx, eax, offset bytesRead, 0
popad
ret
procReadKey endp
; -----------------------------------------------------------------------------
; -----------------------------------------------------------------------------
procSetTitle proc
pushad
cld;
INVOKE SetConsoleTitle, edx;
popad;
ret
procSetTitle endp;
; -----------------------------------------------------------------------------
; -----------------------------------------------------------------------------
main PROC
;; Main procedure. Just initializes stdout, dumps the string, and exits.
; -----------------------------------------------------------------------------
mov edx, offset consoleTitle;
INVOKE procSetTitle;

INVOKE GetStdHandle, STD_OUTPUT_HANDLE ; use Win32 to put
; stdout handle in EAX
mov [consoleOutHandle],eax ; Put the address of the handle in
; our variable

mov edx,offset message ; load the address of the message
; into edx for procWriteString
INVOKE procWriteString ; invoke our write string method.
; -czekamy na nacisniecie jednego klawisza.
invoke GetStdHandle, STD_INPUT_HANDLE ;
mov [consoleInHandle],eax ;
mov edx, offset buffer;
INVOKE procReadKey;

INVOKE ExitProcess,0 ; Windows method to quit
main ENDP
; -----------------------------------------------------------------------------
END main
```

4. Building and debugging code

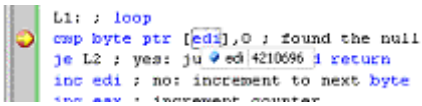
a. Build solution

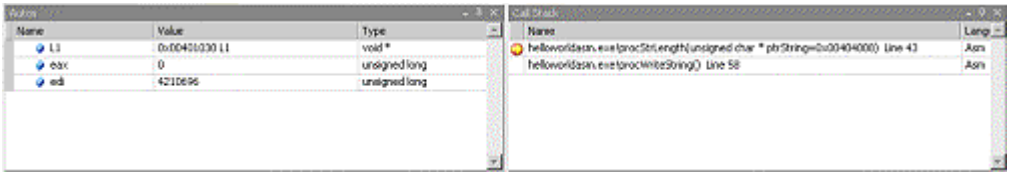
Building a solution is as easy as pressing F7 key or choosing [Menu]/Build/Build Solution. Right after there should be a result information available in Output Window. You may jump directly to the error line (if any) within source code by double clicking a suitable message in Output Window:



b. Debug code

As far as your project is configured accordingly to the rules presented above, you may debug your source code. Simply press F10 or choose [Menu]/Debug/Step… command. During debug you may set breakpoints as well as evaluate CPU registers, variables, call stack, procedure arguments, etc.:





5. Tasks to perform during laboratory.

a. Prepare programming environment using this instruction.
b. Run sample program.
c. Modify sample program so it has console name “Program testowy” instead of path, also modify program so it analyses no more than 10 letters after displaying “Hello world” and writes again input text with lowercase letters converted to uppercase.