 Politechnika Śląska	Instytut Informatyki Politechniki Śląskiej	
Rodzaj studiów SS/NSI/NSM	Języki Asemblerowe	LAB 1

TEMAT:

Tworzenie projektu asemblerowego dla środowiska Visual Studio 2017 i nowszym.

CEL:

Celem ćwiczenia jest poznanie możliwości VS w zakresie tworzenia i uruchamiania aplikacji z kodem mieszanym w języku C++ oraz asemblerze. W założeniu aplikacja składa się z dwóch elementów – aplikacji napisanej w j. C++ oraz biblioteki DLL napisanej w asemblerze dla środowiska Windows. Konstrukcja projektu zakłada możliwość wywoływania funkcji bibliotecznych napisanych w asemblerze z poziomu aplikacji oraz pokazuje prawidłową konfigurację środowiska umożliwiającą debugowanie kodu do poziomu asemblera, obserwację stanu rejestrów i flag procesora czy obszarów pamięci danych.

ZAŁOŻENIA:

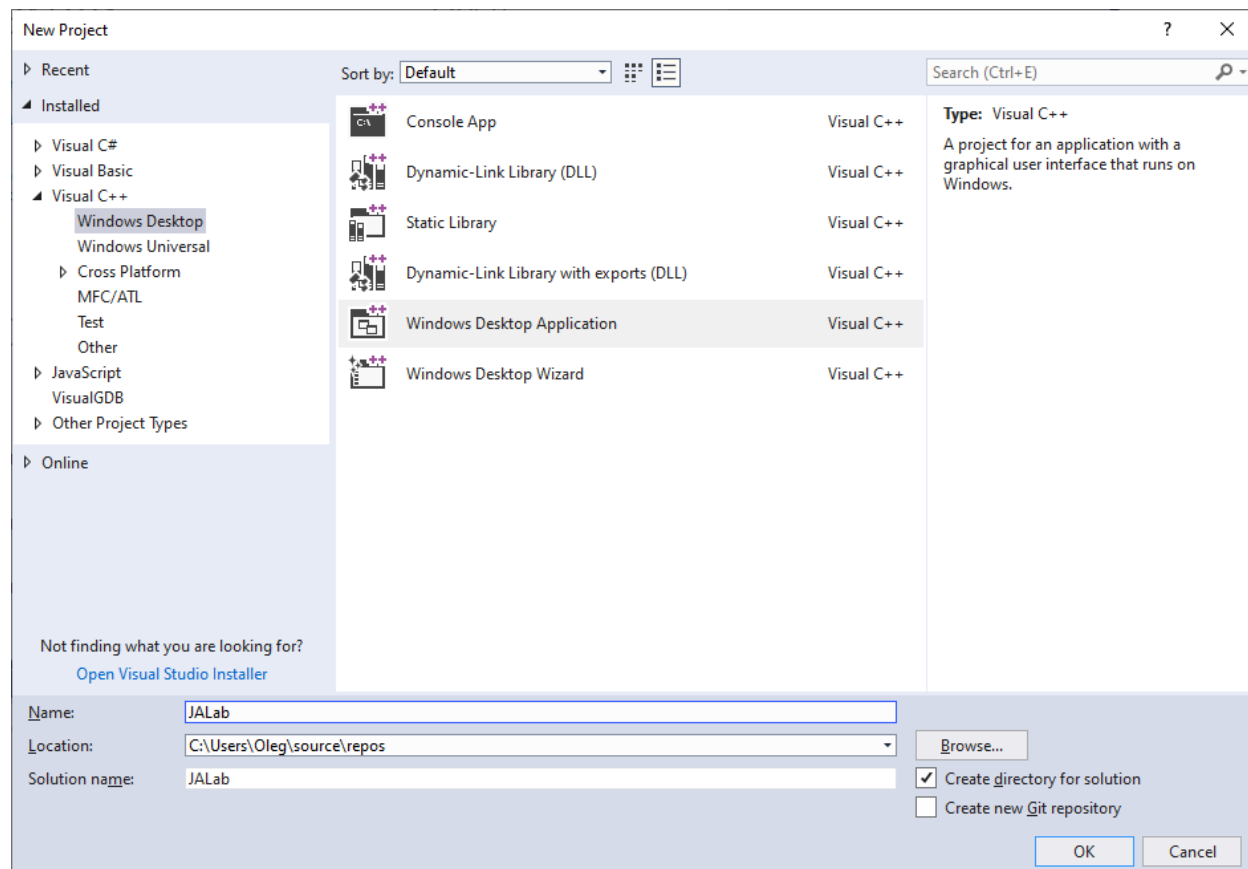
W środowisku VS zakładamy rozwiązanie składające się z dwóch projektów:

- Projekt aplikacja Windows Desktop w j. C++,
- Projekt biblioteka DLL w asemblerze,



W bibliotece DLL utworzona zostanie funkcja asemblerowa, której wywołanie i przekazywanie parametrów wystąpi w aplikacji.

WYKONANIE:

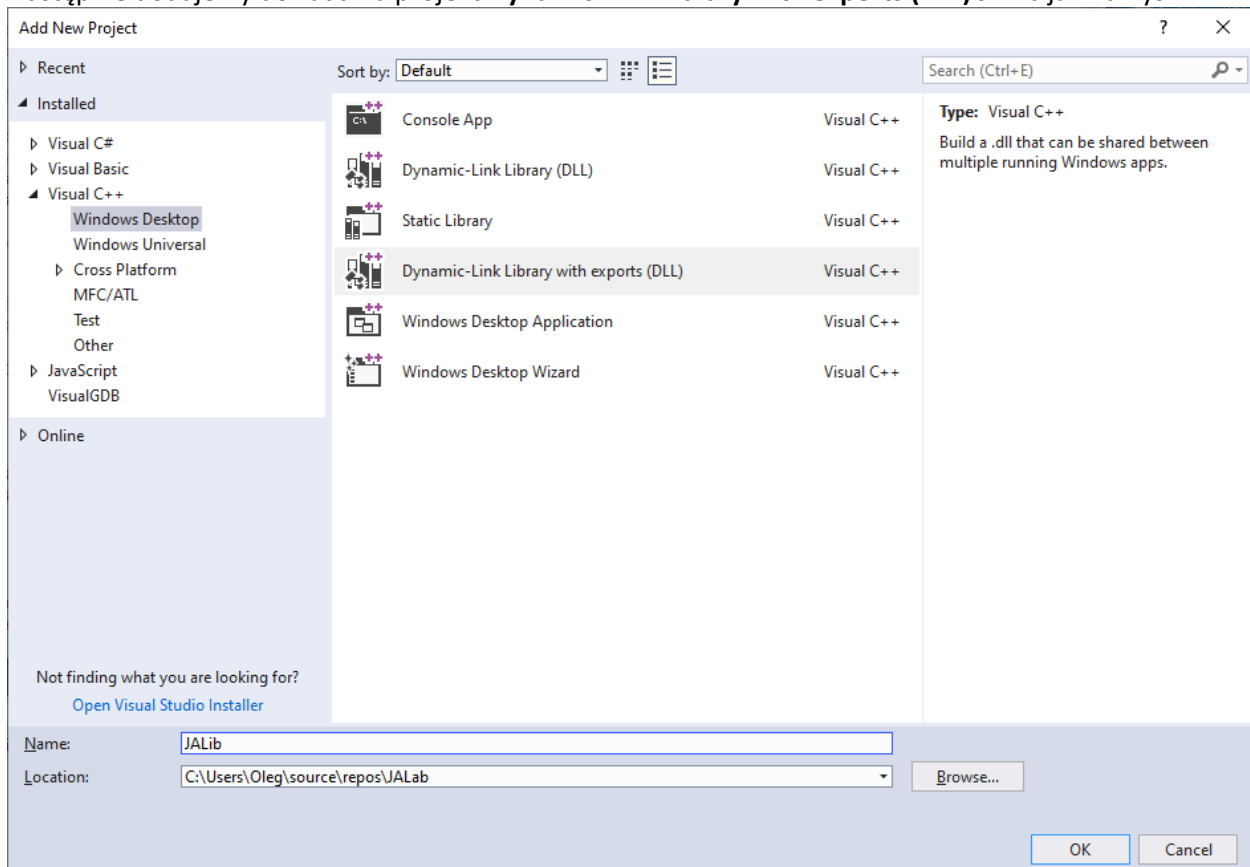
W środowisku VS 2017 lub nowsze tworzymy nowe rozwiązanie nazwie **JALab** wybierając nowy **Windows Desktop Application** o nazwie **JALab** jak na Rys. 1 (lewy) i opcjami (prawy):



Rys. 1 Nowy projekt C++

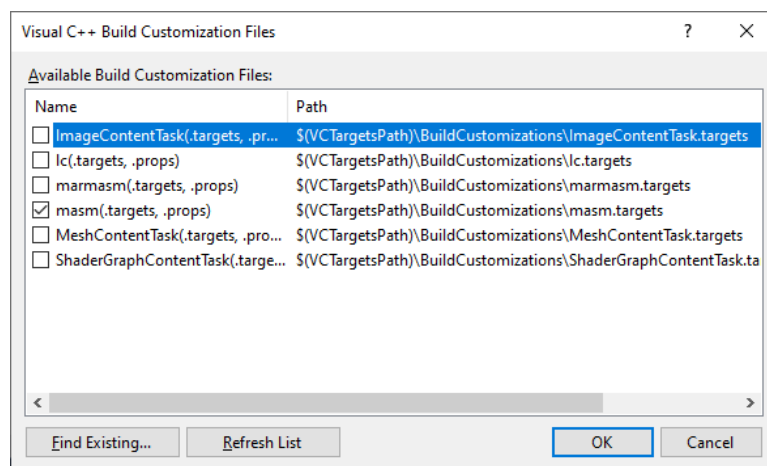
 Politechnika Śląska	Instytut Informatyki Politechniki Śląskiej	
Rodzaj studiów SS/NSI/NSM	Języki Asemblerowe	LAB 1

Następnie dodajemy do zadania projekt **Dynamic-Link Library with exports (DLL) JALib** jak na Rys. 2:





Rys. 2 Nowy projekt DLL

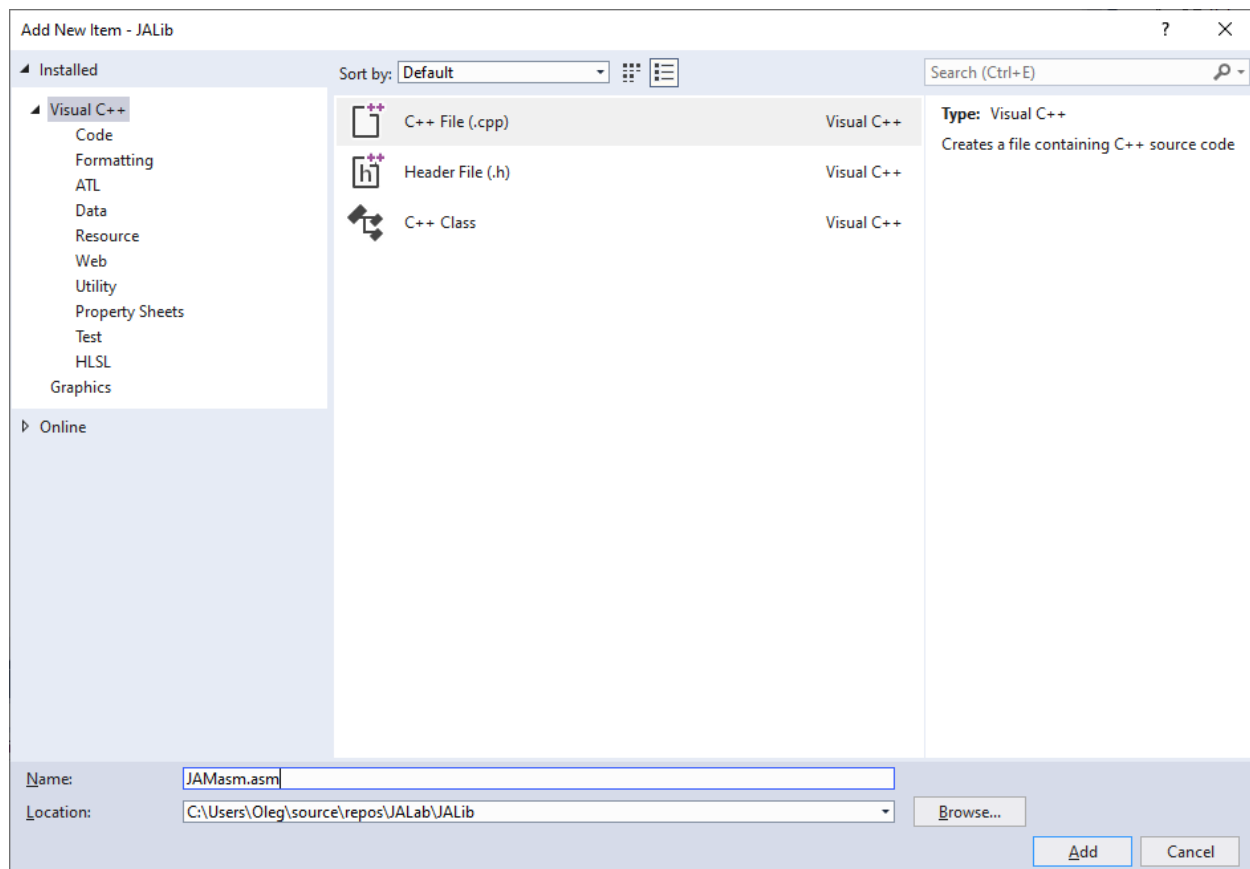
We właściwościach projektu JALib w opcji Menu **Build Dependencies/Build Customizations...** zaznaczamy chęć użycia asemblera **masm** do asemblacji plików z rozszerzeniem ***.asm** przedstawionym na Rys. 3.





Rys. 3 Użycie asemblera **masm** do asemblacji plików ASM

 Politechnika Śląska	Instytut Informatyki Politechniki Śląskiej	
Rodzaj studiów SS/NSI/NSM	Języki Asemblerowe	LAB 1

Wybieramy projekt **JALib** za pomocą myszki, a następnie po wciśnięciu prawego przycisku myszki wybieramy menu *Add/Existing Item....*. Pojawia się okno dialogowe przedstawione na Rys. 4. Wpisujemy nazwę pliku **JAMasm.asm** i dodajemy go do projektu.



Rys. 4 Dodanie pliku JAMasm.asm do projektu JALib

 Politechnika Śląska	Instytut Informatyki Politechniki Śląskiej	
Rodzaj studiów SS/NSI/NSM	Języki Asemblerowe	LAB 1

W pliku **JAMasm.asm** wklejamy ciąg rozkazów makorasemblera X86.

JAMasm.asm:

```
;*****
;*      Laboratory 1 simple assembly procedure call      *
;*      *                                                *
;*      Standard Windows memory model                  *
;*      *                                                *
;*****

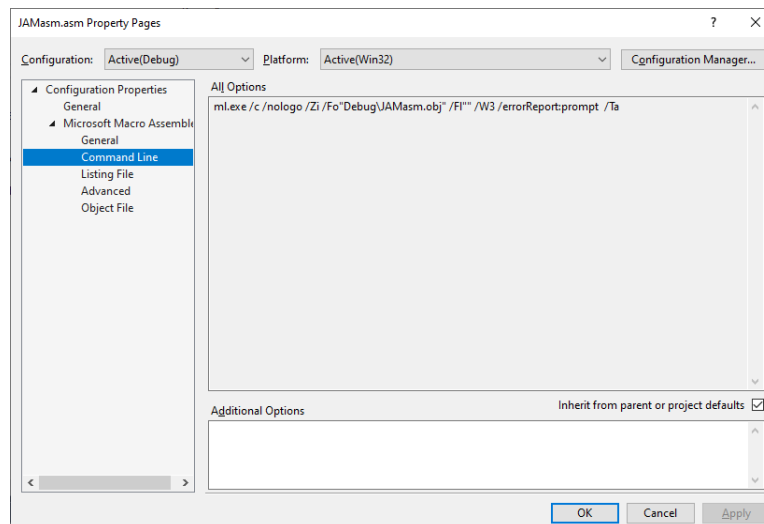
.model flat, stdcall
.code

;*****
;*      *                                                *
;*      Assembler procedure MyProc1 changes Flags register *
;*      *                                                *
;*      *                                                *
;*      Input:  x: DWORD (C++ int type), y: DWORD (C++ int type) *
;*      Output: z: DWORD (C++ int type) in the EAX register *
;*      *                                                *
;*****

MyProc1 proc x: DWORD, y: DWORD
    xor eax,eax      ; EAX = 0
    mov eax,x        ; Param1 eax = x
    mov ecx,y        ; Param2 ecx = y
    ror ecx,1        ; shift ecx right by 1
    shld eax,ecx,2    ; set flags registry
    jnc ET1
    mul y
    ret              ; return z in EAX register
ET1: mul x
    neg y
    ret              ; return z in EAX register
MyProc1 endp

end                  ; End of ASM file
```

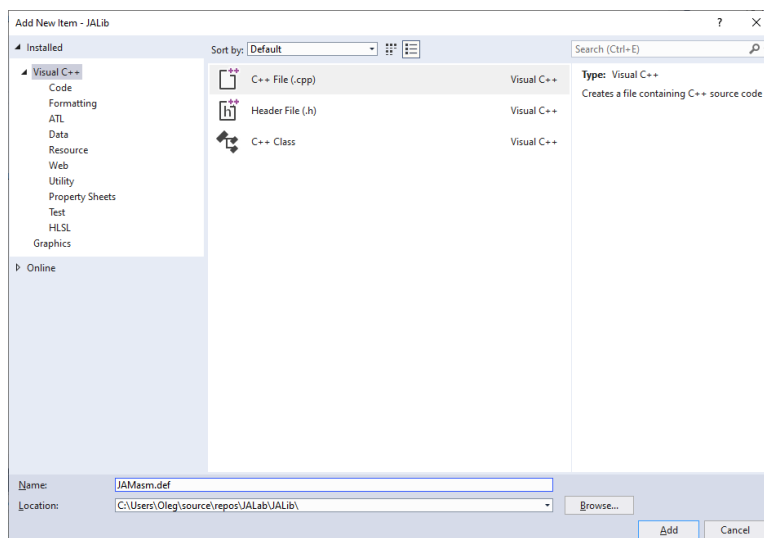
W właściwościach pliku **JAMasm.asm** możemy sprawdzić, że będzie on asemblowany za pomocą makroasemblera VS z opcjami widocznymi na Rys. 5.



 Politechnika Śląska	Instytut Informatyki Politechniki Śląskiej	
Rodzaj studiów SS/NSI/NSM	Języki Asemblerowe	LAB 1

Rys. 5 Właściwości pliku JAMasm.asm

Następnie tworzymy plik definicji eksportów funkcji asemblerowych w bibliotece JALib.dll poprzez utworzenie pliku **JAMasm.def** i dodanie go do *Source Files* w projekcie **JALib**. Operację tę przedstawia Rys. 6.



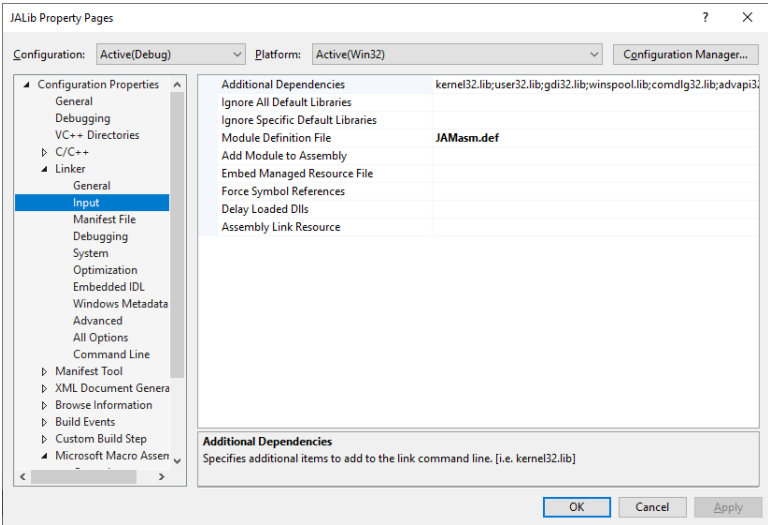
Rys. 6 Tworzenie pliku definicji funkcji asemblerowych w projekcie JALib

W pliku tym wpisujemy definicje eksportowanych nazw funkcji asemblerowych.

JAMasm.def:

```
LIBRARY JALib
EXPORTS
MyProc1
```

W właściwościach projektu **JALib** w opcji Linker/Input wpisujemy w sekcji Module Definition File nazwę pliku **JAMasm.def** tak jak przedstawia to Rys. 7



Rys. 7 Module Definition File w opcji Linker

Aplikacja **JALab.exe** służy do wywołania funkcji bibliotecznych z biblioteki **JALib** oraz udostępnia Interfejs Użytkownika w środowisku Windows.

W aplikacji **JALab** funkcje biblioteczne można wywoływać na dwa sposoby:

- Dynamicznie ładując bibliotekę **JALib.dll** i wywołując funkcję MyProc1,
- Statycznie poprzez linkowanie pliku **JALib.lib** w opcjach linkera aplikacji **JALab**.

W tym celu w pliku programu głównego należy wstawić odpowiednie wywołania funkcji MyProc1.

JALab.cpp:

```
// JALab.cpp : Defines the entry point for the application.
```

```
#include "framework.h"
#include "JALab.h"
```

```
#define MAX_LOADSTRING 100
```



```
// Global Variables:
HINSTANCE hInst; // current instance
WCHAR szTitle[MAX_LOADSTRING]; // The title bar text
WCHAR szWindowClass[MAX_LOADSTRING]; // the main window class name
typedef HRESULT(CALLBACK* LPFNLLFUNC)(UINT, UINT); // DLL function handler
```

```
// Forward declarations of functions included in this code module:
```

```
ATOM MyRegisterClass(HINSTANCE hInstance);
BOOL InitInstance(HINSTANCE, int);
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);
```

```
int WINAPI wWinMain(_In_ HINSTANCE hInstance, _In_opt_ HINSTANCE hPrevInstance, _In_ LPWSTR lpCmdLine,
_In_int nCmdShow) {
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);
```

```
// TODO: Place code here.
```

 Politechnika Śląska	Instytut Informatyki Politechniki Śląskiej	
Rodzaj studiów SS/NSI/NSM	Języki Asemblerowe	LAB 1

```
// Initialize global strings
LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
LoadStringW(hInstance, IDC_JALAB, szWindowClass, MAX_LOADSTRING);
MyRegisterClass(hInstance);

/*****
// Call the MyProc1 assembler procedure from the JALib.dll library in static mode

int x = 3, y = 4, z = 0;
z = MyProc1(x, y); // Call MyProc1 from the JALib.dll library in static mode

HINSTANCE hDLL = LoadLibrary(L"JALib"); // Load JALib.dll library dynamically
LPFNDLLFUNC lpfnDllFunc1; // Function pointer

x = 3, y = 4, z = 0;
if (NULL != hDLL) {
    lpfnDllFunc1 = (LPFNDLLFUNC)GetProcAddress(hDLL, "MyProc1");
    if (NULL != lpfnDllFunc1) {
        z = lpfnDllFunc1(x, y); // Call MyProc1 from the JALib.dll library dynamically
    }
}
*****/

// Perform application initialization:
if (!InitInstance(hInstance, nCmdShow)) {
    return FALSE;
}
HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_JALAB));

MSG msg;



// Main message loop:
while (GetMessage(&msg, nullptr, 0, 0)) {
    if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
}
return (int)msg.wParam;
}

// PURPOSE: Registers the window class.
ATOM MyRegisterClass(HINSTANCE hInstance) {
    WNDCLASSEXW wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);
    wcex.style = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc = WndProc;
    wcex.cbClsExtra = 0;
    wcex.cbWndExtra = 0;
    wcex.hInstance = hInstance;
    wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_JALAB));
    wcex.hCursor = LoadCursor(nullptr, IDC_ARROW);
    wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
    wcex.lpszMenuName = MAKEINTRESOURCEW(IDC_JALAB);
    wcex.lpszClassName = szWindowClass;
    wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));

    return RegisterClassExW(&wcex);
}

// FUNCTION: InitInstance(HINSTANCE, int)
// PURPOSE: Saves instance handle and creates main window
// COMMENTS:
```

 Politechnika Śląska	Instytut Informatyki Politechniki Śląskiej	
Rodzaj studiów SS/NSI/NSM	Języki Asemblerowe	LAB 1

```
// In this function, we save the instance handle in a global variable and
// create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow) {
    hInst = hInstance; // Store instance handle in our global variable

    HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);



    if (!hWnd) {
        return FALSE;
    }
    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    return TRUE;
}

// FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)
// PURPOSE: Processes messages for the main window.
//
// WM_COMMAND - process the application menu
// WM_PAINT - Paint the main window
// WM_DESTROY - post a quit message and return
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam) {
    switch (message) {
        case WM_COMMAND: {
            int wmId = LOWORD(wParam);
            // Parse the menu selections:
            switch (wmId) {
                case IDM_ABOUT:
                    DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
                    break;
                case IDM_EXIT:
                    DestroyWindow(hWnd);
                    break;
                default:
                    return DefWindowProc(hWnd, message, wParam, lParam);
            }
        }
        break;
        case WM_PAINT: {
            PAINTSTRUCT ps;
            HDC hdc = BeginPaint(hWnd, &ps);
            // TODO: Add any drawing code that uses hdc here...

            EndPaint(hWnd, &ps);
        }
        break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}

// Message handler for about box.
INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam) {
    UNREFERENCED_PARAMETER(lParam);
    switch (message) {
        case WM_INITDIALOG:
            return (INT_PTR)TRUE;
    }
}
```


 Politechnika Śląska	Instytut Informatyki Politechniki Śląskiej	
<i>Rodzaj studiów SS/NSI/NSM</i>	<i>Języki Asemblerowe</i>	<i>LAB 1</i>

```

case WM_COMMAND:
if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL) {
    EndDialog(hDlg, LOWORD(wParam));
    return (INT_PTR)TRUE;
}
break;
}
return (INT_PTR)FALSE;
}

```

Następnie w pliku **JALab.h** należy umieścić prototyp funkcji MyProc1 eksportowanej z biblioteki **JALib.dll**:

JALab.h:

```



#pragma once

#include "resource.h"

// external JALib.dll library function definition prototype
extern "C" int _stdcall MyProc1(DWORD x, DWORD y);

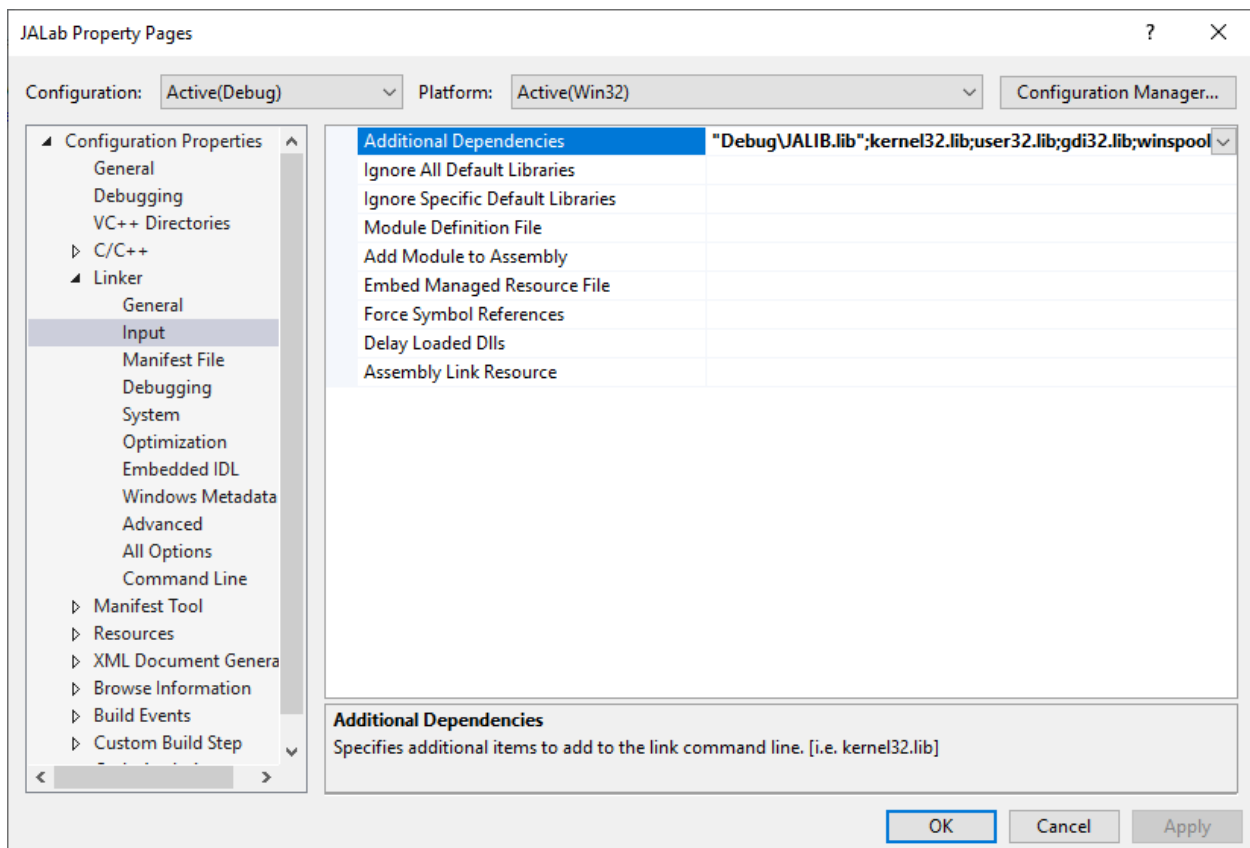
```

Po wykonaniu tych czynności w opcjach projektu **JALab/Linker/Input** należy zmodyfikować pole *Additional Dependencies*: wstawiając nazwę biblioteki **JALIB.lib** w celu umożliwienia statycznego wywołania funkcji *MyProc1*.



 Politechnika Śląska	Instytut Informatyki Politechniki Śląskiej	
Rodzaj studiów SS/NSI/NSM	Języki Asemblerowe	LAB 1

Należy zwrócić uwagę, że plik JALIB.lib tworzony jest w procesie kompilacji projektu **JALib** w katalogu tego projektu w podkatalogu *Debug* dlatego w *Additional Dependencies* należy podać nazwę **JALIB.lib** z uwzględnieniem ścieżki do *Debug* tak jak pokazano to na Rys.8:

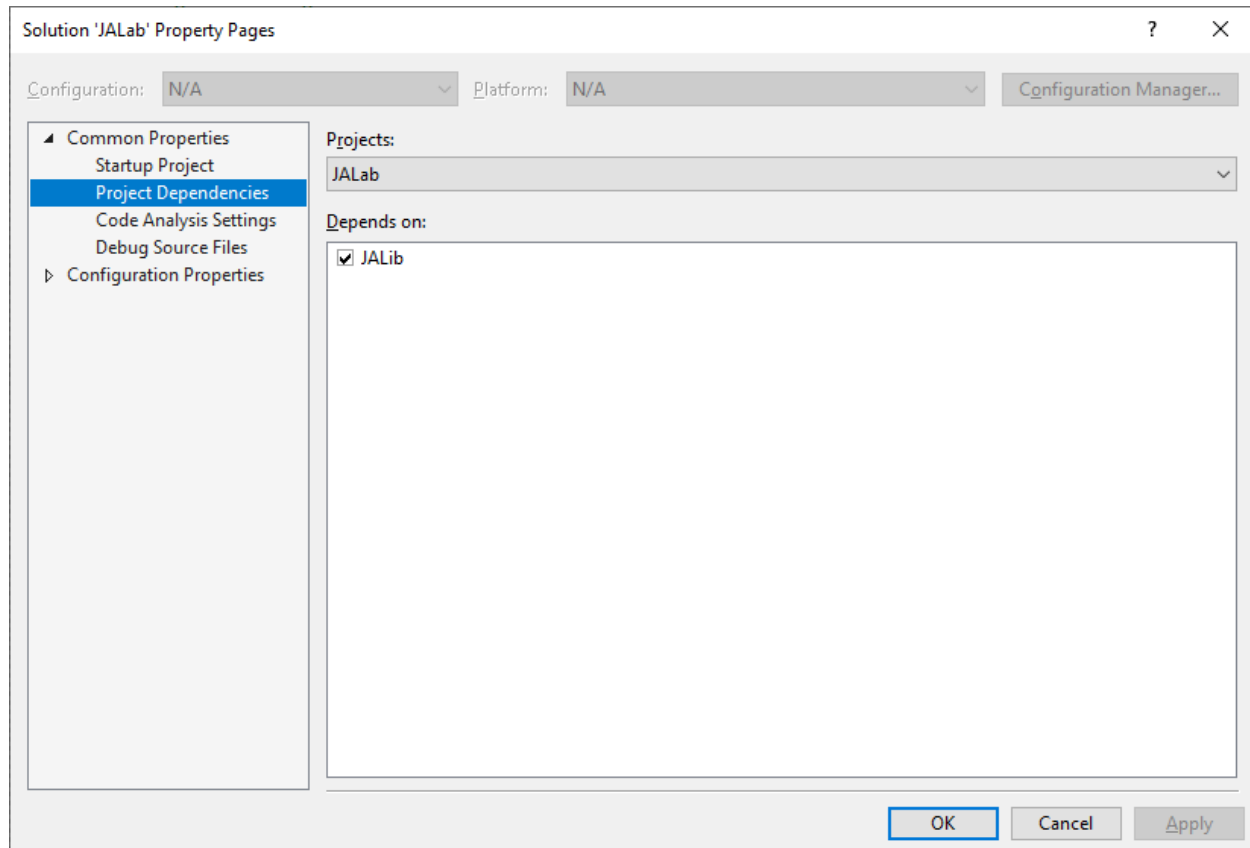
"Debug\JALIB.lib";kernel32.lib;user32.lib;gdi32.lib;winspool.lib;comdlg32.lib;advapi32.lib;shell32.lib;ole32.lib;oleaut32.lib;uuid.lib;odbc32.lib;odbccp32.lib;%(AdditionalDependencies)





Rys. 8 Opcje Linkera dla biblioteki JALib.lib

 Politechnika Śląska	Instytut Informatyki Politechniki Śląskiej	
Rodzaj studiów SS/NSI/NSM	Języki Asemblerowe	LAB 1

Po wykonaniu powyższych czynności należy jeszcze ustawić kolejność kompilacji projektów **JALab** i **JALib** w rozwiązaniu **JALab**. Kolejność kompilacji można ustawić w opcji *Project Dependencies* (Rys. 9)





Rys. 9 Kolejność kompilacji modułów rozwiązania JALab

 Politechnika Śląska	Instytut Informatyki Politechniki Śląskiej	
<i>Rodzaj studiów SS/NSI/NSM</i>	<i>Języki Asemblerowe</i>	<i>LAB 1</i>

PROJEKT W WINDOWS FORMS APPLICATIONS.

W przypadku użycia jako aplikacji wywołującej *Windows Forms Application* istnieje konieczność modyfikacji standardowych parametrów linkera, aby możliwe było debugowanie krokowe.

1. Zaczynamy od utworzenia nowego projektu:
File -> New -> Project
W eksploratorze wybieramy **Visual C++ -> CLR -> Windows Forms Application** i podajemy nazwę **JALab**.
2. Następnie dodajemy projekt **JALib** tak jak opisano powyżej.
3. Teraz klikamy PPM na **Solutions** (eksplorator z lewej strony ekranu) i wybieramy z końca **Properties** po czym wybieramy **Project Dependencies** i w rozwijalnym menu wybieramy **JALab** oraz zaznaczamy poniżej **biblioteka**. Zabieg ten służy ustaleniu zależności pomiędzy projektami.
4. Następnie klikamy PPM na **JALab** (i znów eksplorator z lewej strony ekranu) i wybieramy z końca **Properties** i wybieramy **Configuration Properties -> General**, a następnie w oknie **Common Language** zmieniamy na z **(/clr :pure)** na **(/clr)**.
5. Następnie w **Configuration Properties -> Linker -> Input** w oknie wybieramy trzy kropki (podanie ścieżki) w **Additional Dependencies** i podajemy następującą ścieżkę:
..\Debug\JALib.lib co potwierdzamy poprzez **OK**.

 Politechnika Śląska	Instytut Informatyki Politechniki Śląskiej	
<i>Rodzaj studiów SS/NSI/NSM</i>	<i>Języki Asemblerowe</i>	<i>LAB 1</i>

ZADANIE

1. Utworzyć rozwiązanie **JALab** wraz z projektami **JALab** oraz **JALib** (wg opisu powyżej). Po sprawdzeniu poprawności działania poprzez ustawienie breakpointa na pierwszym rozkazie procedury *MyProc1*, uruchomić program (*Run*) i zaobserwować, że debugger zatrzymuje się prawidłowo na rozkazie:

```
xor  eax, eax          ; EAX = 0
```

wyświetlając poprawnie kod źródłowy pliku **JAMasm.asm**.

2. Zmodyfikować procedurę **MyProc1** tak aby można było wykonując ją krokowo zaobserwować zmiany znaczników rejestru flag: OV UP EI PL ZR AC PE CY
3. Wygenerować indywidualne sprawozdanie w formacie PDF zawierające zrzut ekranowy okna debuggera w trakcie wykonywania rozkazów asemblera.