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

## TEMAT:

Tworzenie projektu asemblerowego dla środowiska Visual Studio 2022.

## 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:

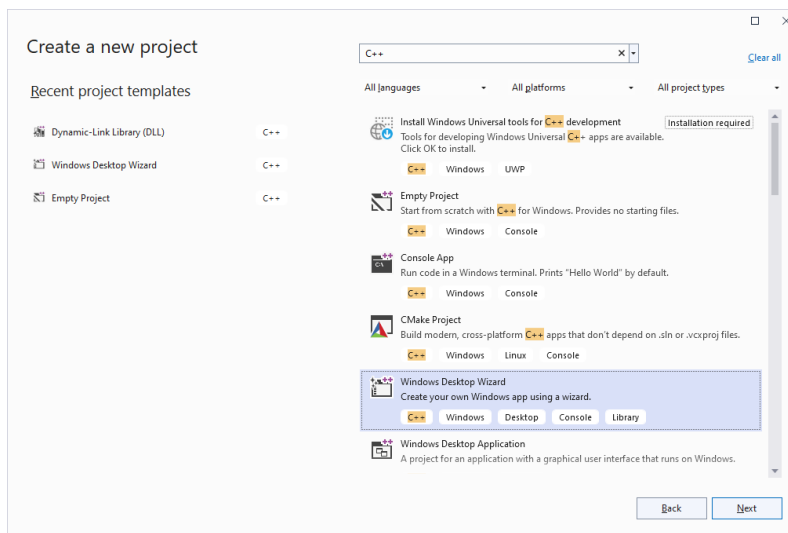
W środowisku VS zakładamy rozwiązanie składające się z dwóch projektów:

- 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.

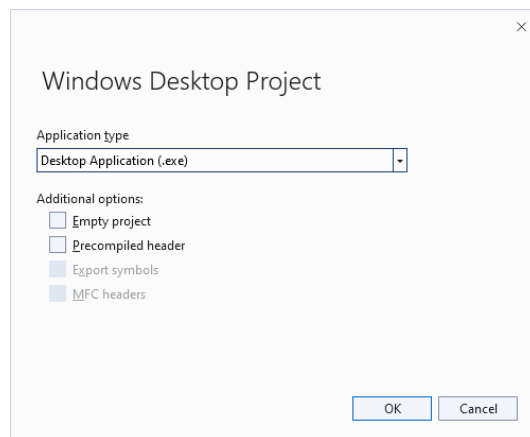
## WYKONANIE:

W środowisku VS 2022 tworzymy nowe rozwiązanie nazwie **JALab1** wybierając nowy **Windows Desktop Application** o nazwie **JALab1** jak na Rys. 1 i 2:



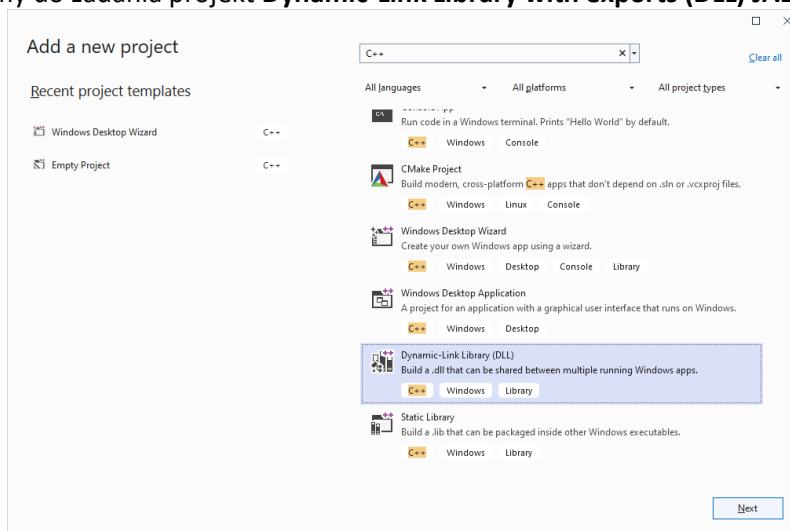
Rys. 1 Nowy projekt C++ JALab1

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



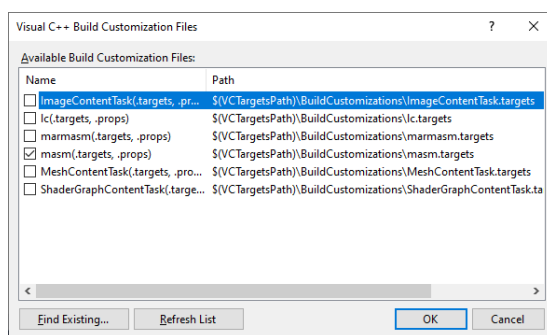
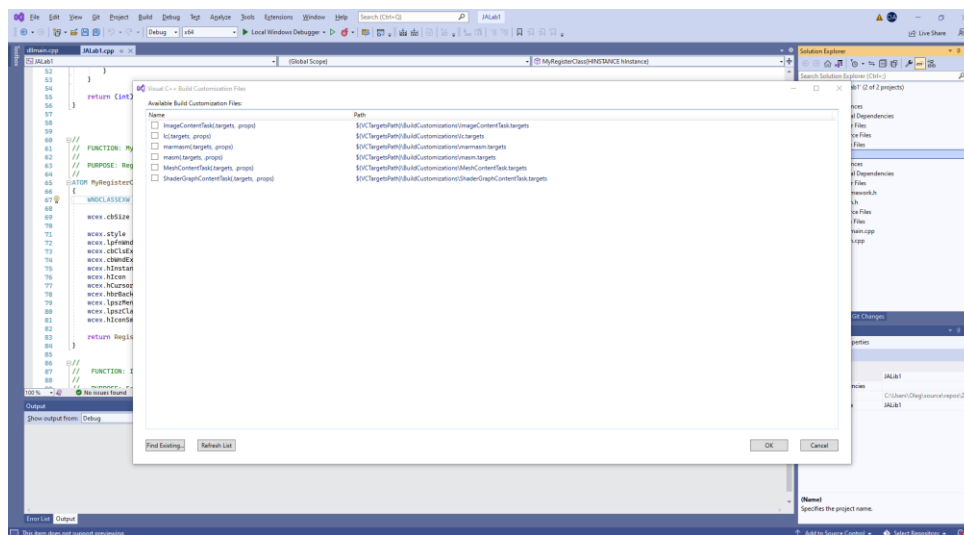
*Rys. 2 Typ projektu C++*

Następnie dodajemy do zadania projekt **Dynamic-Link Library with exports (DLL) JALib1** jak na Rys. 3:



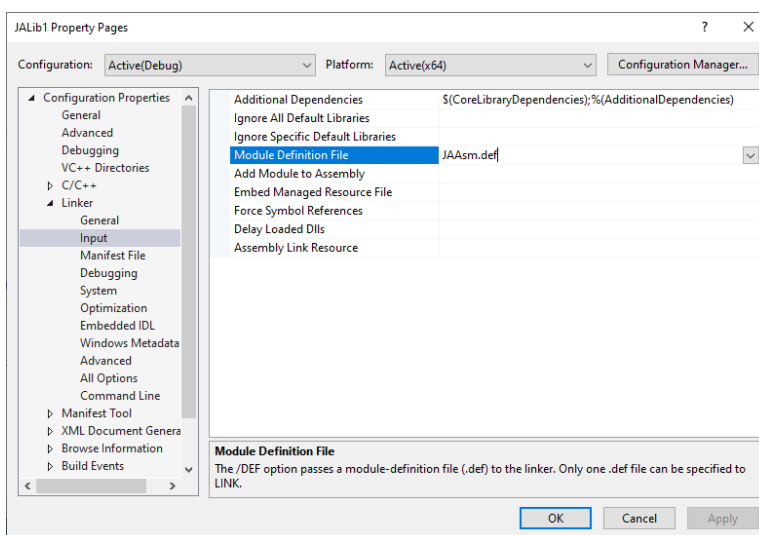
*Rys. 3 Wybór biblioteki DLL dla projektu C++*

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. 4.



Rys. 3 Wybór makroassemblera dla DLL projektu C++

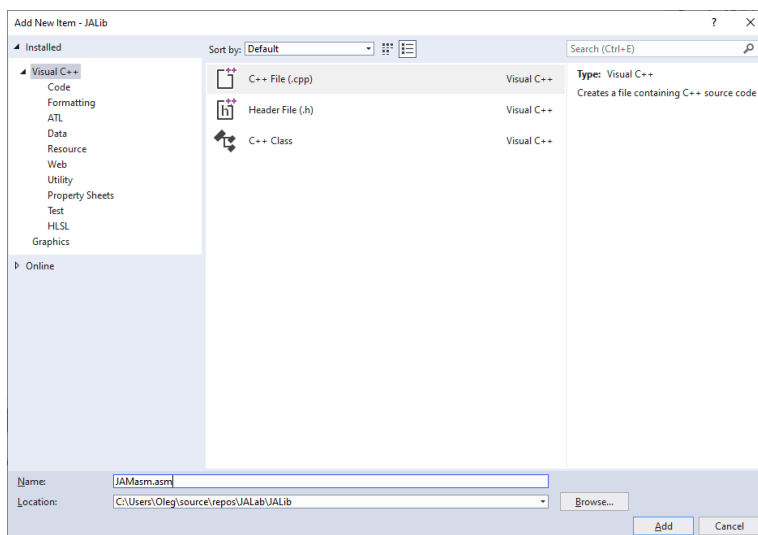
W konfiguracji Linkera biblioteki DLL wpisujemy nazwę pliku \*.def zawierającego eksporty funkcji asemblerowych do linkowania z DLL Rys. 4.



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

Rys. 4 Wybór pliku \*.def dla DLL projektu C++

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. 5. Wpisujemy nazwę pliku **JAMasm.asm** i dodajemy go do projektu.



Rys. 5 Dodanie pliku JAasm.asm do projektu JALib1

W pliku **JAasm.asm** wklejamy ciąg rozkazów makorasemblera X64.

#### **JAasm.asm:**


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

.code

;*****
;*      *                                                                           *
;*      Assembler procedure MyProc1 changes Flags register in X64 mode         *
;*      *                                                                           *
;*      Input:  x: QWORD (C++ int type), y: QWORD (C++ int type)                *
;*      x in RCX, y in RDX                                                         *
;*      Output: z: QWORD (C++ int type) in the RAX register                      *
;*      *                                                                           *
;*****

MyProc1 proc x: QWORD, y: QWORD
    xor rax, rax        ; RAX = 0

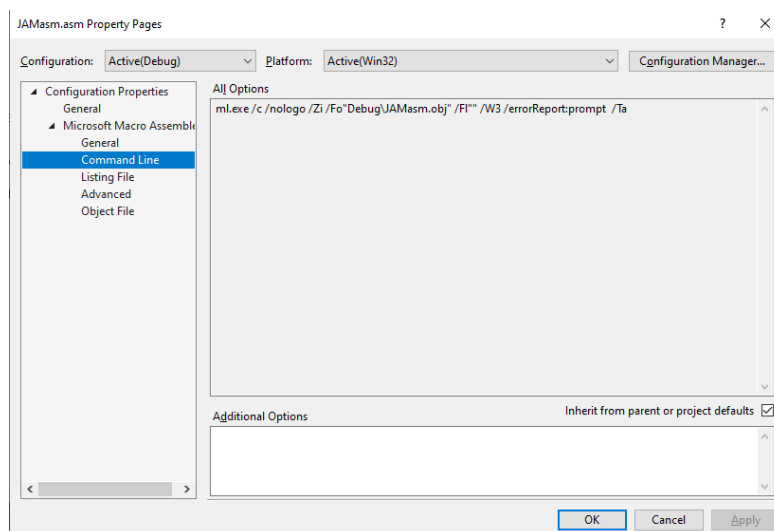
    ror rcx,1           ; shift rcx right by 1
    shld rcx,rcx,2      ; set flags registry
    jnc ET1
    mul y
    ret                 ; return z in RAX register
ET1: mul x
    neg y
    ret                 ; return z in RAX register
```

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

```
MyProc1 endp
```

```
end ; End of ASM file
```

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



Rys. 2 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**. W pliku tym wpisujemy definicje eksportowanych nazw funkcji asemblerowych.

#### JASm.def:

```
LIBRARY JALib1
EXPORTS
MyProc1
```

Aplikacja **JALab1.exe** służy do wywołania funkcji bibliotecznych z biblioteki **JALib1** oraz udostępnia Interfejs Użytkownika w środowisku Windows.

W aplikacji **JALab1** funkcje biblioteczne można wywoływać na dwa sposoby:

- Dynamicznie ładując bibliotekę **JALib1.dll** i wywołując funkcję **MyProc1**,
- Statycznie poprzez linkowanie pliku **JALib1.lib** w opcjach linkera aplikacji **JALab1**.

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



#### JALab1.cpp:

```
// JALab1.cpp : Defines the entry point for the application.

#include "framework.h"
#include "JALab1.h"

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst; // current instance
```

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

```

WCHAR szTitle[MAX_LOADSTRING];           // The title bar text
WCHAR szWindowClass[MAX_LOADSTRING];     // the main window class name

// 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);
typedef HRESULT(CALLBACK* LPFNDLFUNC)(UINT, UINT); // DLL Library function handler

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

    // Initialize global strings
    LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadStringW(hInstance, IDC_JALAB1, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);

    /*****
    // Call the MyProc1 assembler procedure from the JALib1.dll library in static mode
    int  x = 3, y = 4, z = 0;

    z = MyProc1(x, y); // Call MyProc1 from the JALib1.dll library in static mode

    // Call the MyProc1 assembler procedure from the JALib1.dll library in dynamic mode
    HINSTANCE hDLL = LoadLibrary(L"JALib1"); // Load JALib.dll library dynamically
    if (hDLL != NULL)
    {
        LPFNDLFUNC lpfnDllFunc1; // Function pointer

        x = 3, y = 4, z = 0;
        if (NULL != hDLL) {
            lpfnDllFunc1 = (LPFNDLFUNC)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_JALAB1));

    MSG msg;

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

```

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

```

    }
}

return (int) msg.wParam;
}

//
// FUNCTION: MyRegisterClass()
//
// 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_JALAB1));
    wcex.hCursor        = LoadCursor(nullptr, IDC_ARROW);
    wcex.hbrBackground  = (HBRUSH) (COLOR_WINDOW+1);
    wcex.lpszMenuName    = MAKEINTRESOURCEW(IDC_JALAB1);
    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:
//
//         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

```



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

```
// 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;

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



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

Następnie w pliku **JALab1.h** należy umieścić prototyp funkcji **MyProc1** eksportowanej z biblioteki **JALib1.dll**:

**JALab1.h:**

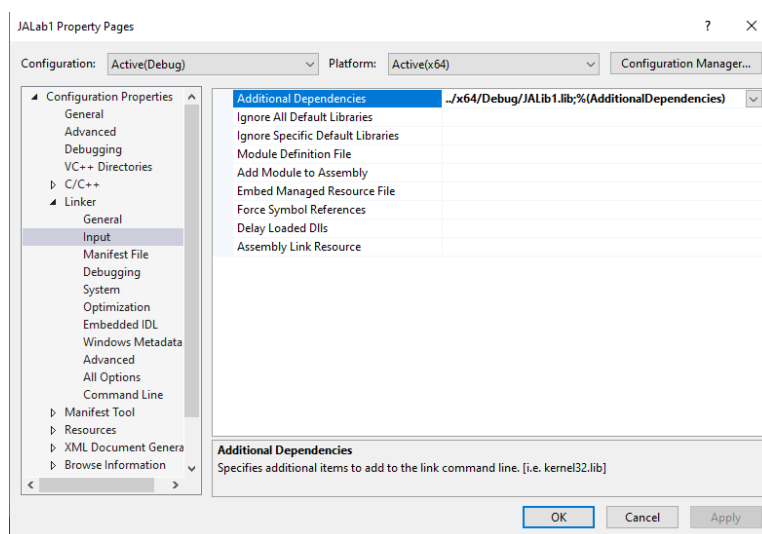
```
#pragma once

#include "resource.h"

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

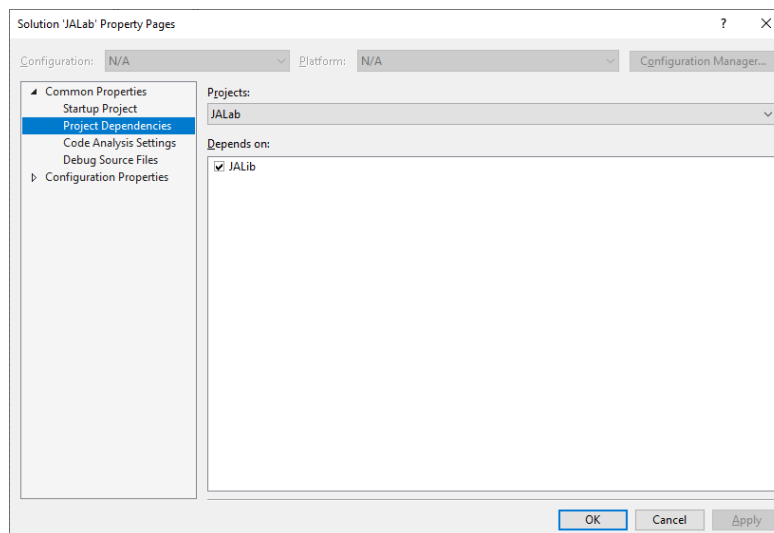
Po wykonaniu tych czynności w opcjach projektu **JALab1/Linker/Input** należy zmodyfikować pole *Additional Dependencies*: wstawiając nazwę biblioteki **JALIB1.lib** w celu umożliwienia statycznego wywołania funkcji **MyProc1**.

Należy zwrócić uwagę, że plik **JALIB1.lib** tworzony jest w procesie kompilacji projektu **JALib1** w katalogu tego projektu w podkatalogu **X64/Debug** dlatego w *Additional Dependencies* należy podać nazwę **JALib1.lib** z uwzględnieniem ścieżki do **..X64/Debug** tak jak pokazano to na Rys.6:



Rys. 6 Opcje Linkera dla biblioteki **JALib1.lib** w projekcie **JALab1**

Po wykonaniu powyższych czynności należy jeszcze ustawić kolejność kompilacji projektów **JALab1** i **JALib1** w rozwiązaniu **JALab1**. Kolejność kompilacji można ustawić w opcji *Project Dependencies* (Rys. 7)





Rys. 7 Kolejność kompilacji modułów rozwiązania JALab1

## 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 **JALab1**.
2. Następnie dodajemy projekt **JALib1** 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 **JALab1** oraz zaznaczamy poniżej **biblioteka**. Zabieg ten służy ustaleniu zależności pomiędzy projektami.
4. Następnie klikamy PPM na **JALab1** (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: **..\X64\Debug\JALib1.lib** co potwierdzamy poprzez **OK**.

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

## 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 rax,rax          ; RAX = 0
```

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

2. Przecwiczyć przekazywanie do procedury *MyProc1* parametrów wg wzoru poniżej. W procedurze *MyProc1* należy wykonać operacje na rejestrach w sposób podobny, jak w przykładzie powyżej wykonując wybrane operacje matematyczne pokazujące prawidłowość przekazania parametrów do procedury ASM:

### Example of argument passing 1 - all integers

```
func1 (int a, int b, int c, int d, int e, int f);
// a in RCX, b in RDX, c in R8, d in R9, f then e pushed on stack
```

### Example of argument passing 2 - all floats

```
func2 (float a, double b, float c, double d, float e, float f);
// a in XMM0, b in XMM1, c in XMM2, d in XMM3, f then e pushed on stack
```

### Example of argument passing 3 - mixed ints and floats

```
func3 (int a, double b, int c, float d, int e, float f);
// a in RCX, b in XMM1, c in R8, d in XMM3, f then e pushed on stack
```

### Example of argument passing 4 - \_\_m64, \_\_m128, and aggregates

```
func4(__m64 a, __m128 b, struct c, float d, __m128 e, __m128 f);
// a in RCX, ptr to b in RDX, ptr to c in R8, d in XMM3,
// ptr to f pushed on stack, then ptr to e pushed on stack
```

3. Wygenerować indywidualne sprawozdanie w formacie PDF zawierające zrzut ekranowy okna debuggera w trakcie wykonywania rozkazów asemblera oraz opisać wyniki działania wywołania procedury *MyProc1*.