Dodawanie biblioteki asm do projektu w cpp

1. Instrukcja jest oparta na instrukcji z JA lab 1. Oryginał przedstawiał jak stworzyć projekt cpp który wywoła funkcję w asm. Instrukcja jest załączona w folderze pod nazwą „Tworzenie projektu cpp z biblioteka asm”
2. Zmieniona instrukcja pokazująca jak dodać bibliotekę asm do zwykłego projektu w cpp. Najlepiej zrobić to na pustym projekcie w którym nie ma innej logiki. Zalecany typ projektu w Visual Studio 2019 to Console App w CPP.
3. Wszystkie odniesienia do „JALab” oznaczają bibliotekę cpp, jej nazwą należy zastąpić wcześniej podaną. Wszystkie odniesienia do „JALib” oznaczają bibliotekę asm i jej składowe, zaleca się by tą część pozostawić bez zmian w nazwie.
4. Można również zaimportować projekt JALib podany w folderze, w tedy odpada cała związana z nim konfiguracja ale cały czas pozostaje ta związana z JALab
5. W folderze podano również przykładowy działający projekt pod nazwą „Call asm from cpp”
6. Przy zmianie z x86 na x64 należy
   1. W properties JALib skopiować wszystkie ustawienie z 32 na 64 (plik JAMasm.def oraz JALib.lib)
   2. Po kompilacji samego pliku asm wrzucić dll i lib do folderu z cpp
   3. Poprawić wymagane różnice w kodzie przedstawione niżej
7. Instrukcja:

W pierwszej kolejności w Środowisku Visual Studio utworzyć projekt Console App w Cpp i przejść dalej.

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

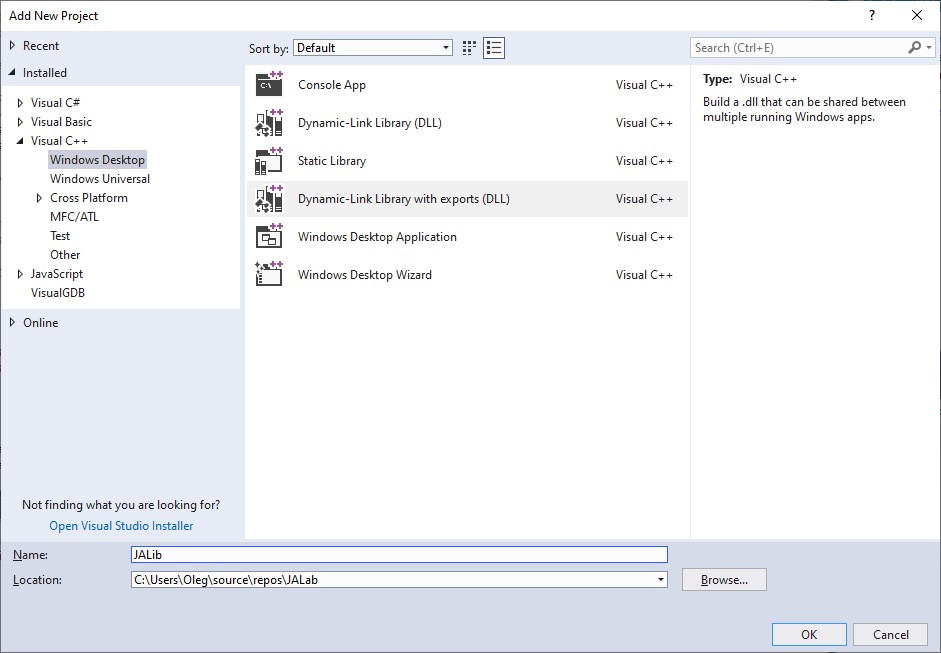
By dodać projekt do istniejącego projektu należy nacisnąć PPM na Solution oraz:(Rys. 1):

Obraz zawierający tekst

Opis wygenerowany automatycznie

*Rys. 1 Dodawanie projektu od istniejącego perojektu*

Potem



*Rys. 2 Nowy projekt DLL*

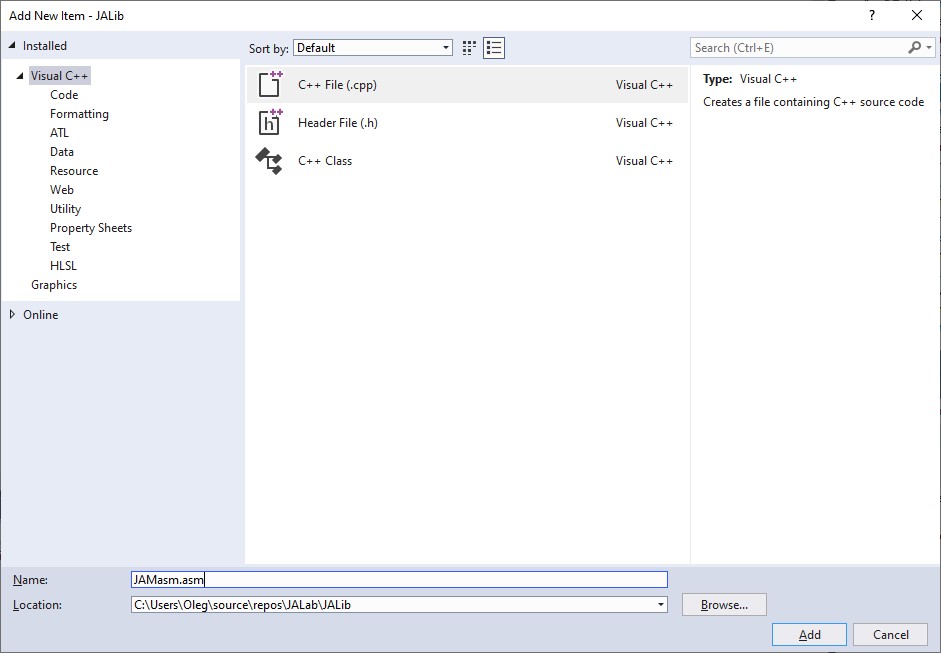
We właściwościach projektu JALib w opcji Menu **Build Dependencies/Build Customizations…** zaznaczamy chęć użycia asemblera **masm** do asemblacji plików z rozszerzeniem **\*.asm** przedstawionym na Rys. 3.

Obraz zawierający tekst

Opis wygenerowany automatycznie

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

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*

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

TO DZIAŁA TYLKO DLA X86. W wersji 32 można bez problemu korzystać ze zmienny przekazanych (Niżej wersja dla 64).

***JAMasm.asm:***

DWORD oznacza zmienną 4 bajtową (32 bitową) i zawsze wymaga stosowanie rejestrów z „e” czyli eax ,ebx ,ecx ,edx itp. Podany program sumuje przekazane dwie liczby i zwraca wartość.

.model flat, stdcall

.code

MyProc1 proc x: DWORD, y: DWORD

MOV eax, x ;Param1 eax = x

ADD eax, y ;Adds y to eax

ret ; return z in EAX register

MyProc1 endp

end ; End of ASM file

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

W wersji dla x64 trzeba usunąć linijkę „.model flat, stdcall” bo się jej nie używa oraz zmieniają się zasady przekazywania parametrów. Pierwsze trzy przekazywane są w rejestrach 1 – rdx, 2 – r8, 3 – r9, a reszta przez stos. Zaleca się ostrożność przy przekazywaniu danych przez rejestry bo lubią nie działać, najlepiej przekazywać dane przez tablice. W kodzie asemblera nie ma różnicy podczas deklaracji czy przekazuje się wartość czy adres. Tu w 1 przekazana jest długość tablicy a w b adres pierwszego elementu. Program sumuje po kolei jej zawartość i zwraca. QWORD oznacza największy typ danych 8 bajtów (64 bity) i trzeba do niego zawsze użyć rejestru z „r” rax, rbi, rci, rdi itp.

;.model flat, stdcall

.code

MyProc1 proc a: QWORD, b: QWORD

MOV rax, [rdx]

ADD rax,[rdx+4]

ADD rax,[rdx+8]

ret

MyProc1 endp

end ; End of ASM file

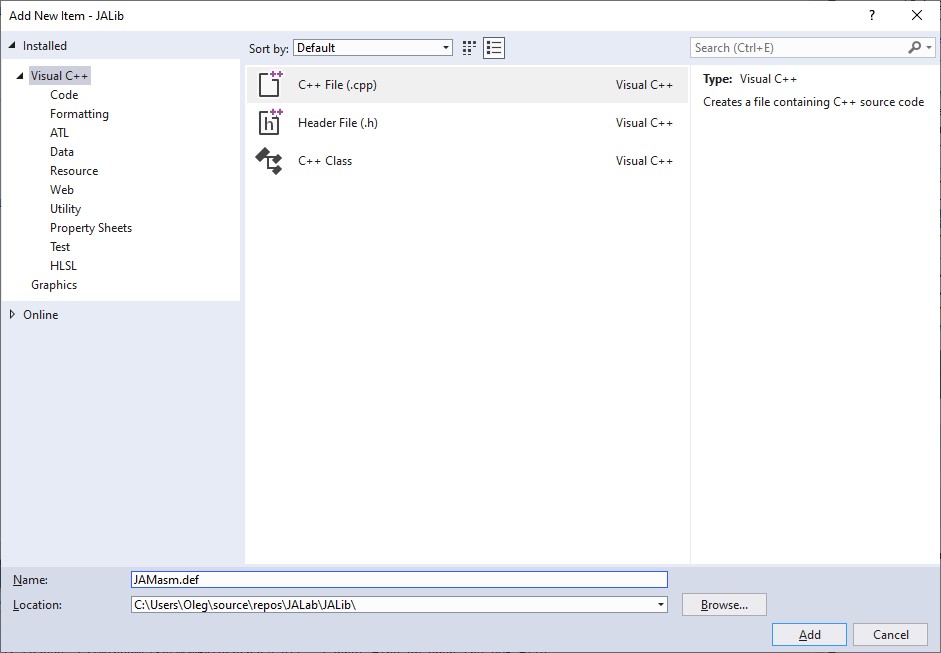
W właściwościach pliku ***JAMasm.asm*** możemy sprawdzić, że będzie on asemblowany za pomocą makroasemblera VS z opcjami widocznymi na Rys. 5.

Obraz zawierający tekst

Opis wygenerowany automatycznie

*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

Obraz zawierający tekst

Opis wygenerowany automatycznie

*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.

UWAGA DLA PRZYKŁADU PROGRAMÓW X86 i 64 są róże funkcji w cpp

***JALab.cpp dla X86:***

#include <windows.h>

#include <iostream>

extern "C" int \_stdcall MyProc1(DWORD x, DWORD y);

int main()

{

std::cout << "Start" << std::endl << std::endl;

int x = 3, y = 4, z = 0;

//Ładuje funkcje MyProc1 z biblioteki JALib.dll napisanej w asm w sposób statyczny

//Funkcja przykładowa pobierała dwie zmienne int i zwracała ich sumę

z = MyProc1(x, y);

std::cout << "Wynik: " << z << std::endl << std::endl;

}

***JALab.cpp dla X64:***

***Jeżeli przekazywana jest tablica to trzeba zaznaczyć wskaźnik w extern deklaracji. Najlepiej przekazywać tablice dyanmiczne***

#include <windows.h>

#include <iostream>

extern "C" int \_stdcall MyProc1(int p1, int\* p2);

int main()

{

int a = 3,z = 0;

int\* tab = new int[3];

tab[0] = 10;

tab[1] = 20;

tab[2] = 30;

//std::cout << "Size x: " << sizeof(x) << std::endl << std::endl;

//Ładuje funkcje MyProc1 z biblioteki JALib.dll napisanej w asm w sposób statyczny

//Funkcja przykładowa pobierała dwie zmienne int i zwracała ich sumę

z = MyProc1(a, tab);

std::cout << "Wynik: " << z << std::endl << std::endl;

}

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

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)

Obraz zawierający tekst

Opis wygenerowany automatycznie

*Rys. 8 Opcje Linkera dla biblioteki JALib.lib*

Jeżeli środowisko nie będzie umiało znaleźć plików to skopiować plik JALib.dll oraz JALib.lib i wkleić do folderu z nazwą głównego projektu (nie JALib) i „Debug/JALib.lib” zmienić na normalne JALib.lib. Jeżeli to też nie zadziała to skopiować te pliki do wszystkich folderów.

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)

Obraz zawierający tekst

Opis wygenerowany automatycznie

*Rys. 9 Kolejność kompilacji modułów rozwiązania JALab*

Zbudować i skompilować program, powinien działać