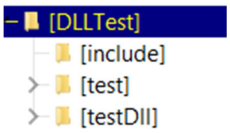
Robimy bibliotekę DLL – na razie nie okienkowo

- Wykreować nowy katalog: `DLLTest`
- Wykreować nowy projekt typu Windows Console Application w tym katalogu (w `DLLTest`) o nazwie **`test`**.
- Kliknąć prawym klawiszem na nazwę `Solution test` i dodać nowy projekt (`Add->New Project`) Windows Desktop typu Dynamic-Link Library (DLL) o nazwie **`testDll`** w katalogu `DLLTest. UWAGA NA LOKALIZACJE!!`
- Oba projekty są w tym samym katalogu `DLLTest`. W katalogu `DLLTest` wykreować dodatkowy podkatalog: include



- Wejść do ustawień projektu głównego **`test`** (w oknie Solution Explorer wybrany projekt **`test`**): Menu Project->ProjectDependecies... Sprawdzić czy projekt **`test`** jest zależny od **`testDll`**. Jeśli pole checkbox nie jest zaznaczone to zaznaczyć. Można też sprawdzić zakładkę Build Order. Powinno być najpierw kompilowane **`testDll`** a potem (poniżej) **`test`.**
- Ponieważ plik nagłówkowy dla dostępu do biblioteki DLL będzie potrzebny również w programie głównym to w projekcie **`testDll`** dodać plik `testDll.h`.
- W ustawieniach projektu **`testDll`** (C/C++->Preprocessor) jest zdefiniowana stała preprocesora TESTDLL_EXPORTS.
- W pliku `testDll.h` wkopiować to co jest w stdafx.h (werjsa 2017) i dodać kod (modyfikują miejsce inkludowania ”targetver.h”):

```
#ifdef TESTDLL_EXPORTS
#define ZDLL_API __declspec(dllexport)
#include "targetver.h"
#else
#define ZDLL_API __declspec(dllimport)
#endif
```

- Można oczywiście użyć innej nazwy niż `ZDLL_API` (będę tej używać tej stałej w opisie) jak również można zmienić nazwę `TESTDLL_EXPORTS` ale wszędzie!
- Całość wygląda tak:

```
#pragma once


#ifdef TESTDLL_EXPORTS
#define ZDLL_API __declspec(dllexport)
#include "targetver.h"
#else
#define ZDLL_API __declspec(dllimport)
#endif

#define WIN32_LEAN_AND_MEAN    // Exclude rarely-used stuff from Windows headers
// Windows Header Files
#include <windows.h>
```

- Plik `testDll.h` ma być inkludowany w pliku `dllmain.cpp` oraz `testDll.cpp`

```
// dllmain.cpp : Defines the entry poir
#include "testDll.h"

BOOL APIENTRY DllMain( HMODULE hModule,
```

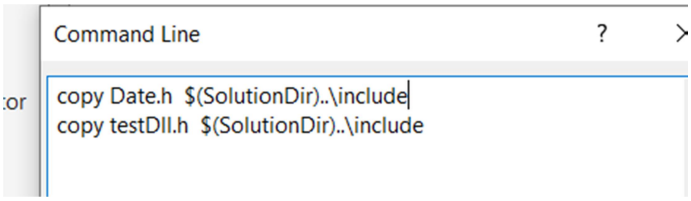
```
// testDll.cpp : Defi
//

#include "testDll.h"
```

- Aby nie było problemów przy kompilacji wyłączyć prekompilowane headery w obu projektach.
- Dodać do projektu **`testDll`** klasę `Time`. Należy od razu (przed implementacją) zainkludować w pliku `Time.cpp` plik `testDll.h` ZAMIAST `Time.h` oraz zainkludować w `testDll.h` plik `Time.h`.
- Teraz można przejść do implementacji tej prostej klasy. Po pierwsze klasa musi być zadeklarowana jako: **`class ZDLL_API Time`** aby była eksportowana do biblioteki tego DLL-a (czyli każda metoda tej klasy z wyjątkiem np. operatorów we/wy bo musza być friend-ami więc są funkcjami globalnymi). Wymagania:

✓ Całkowite składowe prywatne `m_nHour, m_nMin, m_nSec`

- ✓ Konstruktor z parametrami domyślnymi (godzina 0:0:0) oraz drugi konstruktor kopiujący
- ✓ Dla implementacji kopiującego należy zdefiniować (choć tu nie jest to konieczne bo kompilator sam dostarcza operator podstawienia ale gdyby były składowe dynamiczne to niezbędne) operator podstawienia. Konstruktor kopiujący MA wywołać operator podstawienia!
- ✓ Gettery i settery (implementowane jako `inline` w pliku nagłówkowym nie w klasie). Gettery oczywiście nie zmieniają obiektu klasy więc `const`.
- ✓ Dla ułatwienia nie trzeba sprawdzać poprawności godzin minut i sekund ale używać metod set aby tylko w jednym miejscu ewentualnie dodawać taką walidację.
- ✓ Metoda `setTime( int, int, int );` (musi wywołać poszczególne setery)
- ✓ Operatory we/wy. Ponieważ to funkcje globalne a muszą być eksportowane to musi być użyta stała preprocesora `ZDLL_API` przed nagłówkiem (po `frined`)

- Dodać do pliku
- W ustawieniach projektu `testDll` zrobić jeszcze zmianę w grupie `Build Events-> Post-Build Event`. Dodać polecenia kopiujące pliki nagłówkowe do wspólnego katalogu (dla wszystkich bibliotek – może być ich więcej) `DLLTest\include`:



- To oczywiście nie jest konieczne ale w przypadku kilku bibliotek warto zgromadzić pliki nagłówkowe w jednym miejscu Moduły (projekty składowe) mogą korzystać ze wszystkiego. Choć można to zrobić ręcznie. Jednak przy dłuższej pracy nad projektem kiedy może cos się zmienić w plikach nagłówkowych łatwiej aby się same kopiowały.
- Ja robie oprócz katalogu include także lib a w nim podkatalogi Debug i Release i ustawiam w projekcie DLL-a, że do katalogu lib/Debug ma być zapisywany plik .lib w wersji Debug a do lib/Release w wersji Release, W ten sposób pliki .lib wszystkich DLL-i mam w tym

samym miejscu projektu (ale tu robimy prościej – bo trzeba wtedy zmienić jeszcze w ustawieniach projektu gdzie są odpowiednie pliki .lib)

- Teraz program powinien się zbudować poprawnie. W wyniku zbudowania bez błędów kompilacji w katalogu test/Debug będzie plik `testDll.dll` (niezbędny do uruchomienia) oraz `testDll.lib` (niezbędny do budowania całego projektu gdy użyjemy potem klas w test).
- Analogicznie (takie same funkcje) zaimplementować klasę `Date` ze składowymi prywatnymi: `m_nDay, m_nMonth, m_nYear`. Data domyślna do 1/1/1900 (separator'/')
- Następnie skompilować i usunąć błędy (pamiętać o zainkludowaniu do Date.cpp zamiast Date.h pliku testDll.h i do niego zainkludować Date.h)
- Zaimplementować klasę `TimeDate` będącą klasą łączną klas `Time` i `Date`. Wymagania:

✓ Pamiętać o zainkludowaniu `testDLL.h` w `TimeDate.cpp` przed `TimeDate.h`

✓ Konstruktor z 6-cioma parametrami domyślnymi

✓ Konstruktor kopiujący

✓ Konstruktor na bazie obiektu typu `Time` i obiektu typu `Date`

✓ Metoda `SetTimeDate()`

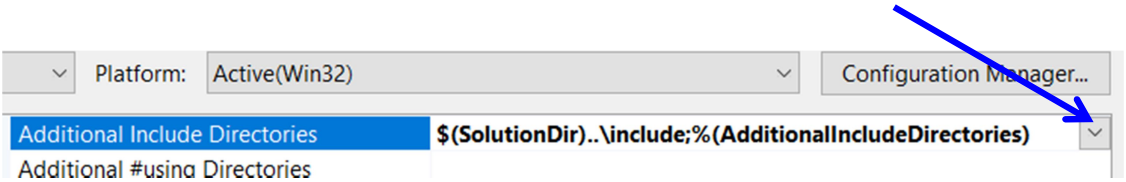
✓ Operator podstawienia (wywoływany z konstruktora kopiującego) – choć przy takich klasach nie potrzeba ale zrobić.

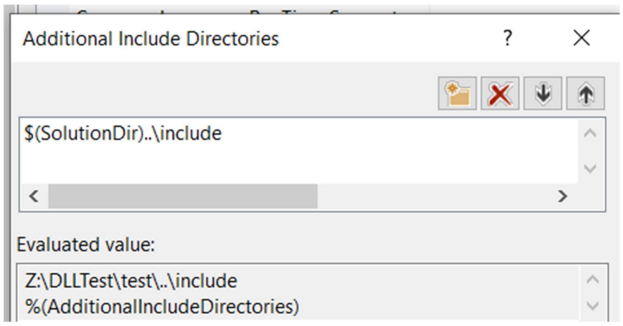
✓ Operaotry we/wy

✓ Implementacja klasy ma się w całości opierać na tym co jest zaimplementowane w klasach bazowych!!!! – wywoływane jedynie odpowienide konstruktory i operaory (żadnych metod z wyjątkiem SetTimeDate())

- Zbudować całość.
- Teraz testowanie. Najpierw trzeba zmodyfikować ustawienia projektu test.

✓ W **C/C++ -> General ->Additional Include Directories** dodać **$(SolutionDir)..\include**; (z prawej jest strzałka ->Edit i otworzy się okineko gdzie można wpisać)





- ✓ W **Linker ->Input** dodać **$(OutDir)testDll.lib**; (z prawej jest strzałka ->Edit i wpisać)
- ✓ Oczywiście jeśli byśmy chcieli też mieć wersje Relese to analogicznie trzeba by zrobić te ustawienia dla tej werjsi.

- Następnie należy zainkludować wprost (zdefiniowana ścieżka) plik ”testDll.h”
- Teraz do main() można wpisać jakiś kod używający klas z DLL-a.

```
#include "testDLL.h"

int main()
{
  Date d( 2020, 5, 15 ); // ja mam date od roku
  cout << "d = " << d << endl;
  Time t( 20, 35, 0 );
  cout << "d = " << d << endl;

  TimeDate td1( t, d );
  TimeDate td2( td1 );

  cout << endl << "td2 = " << td2 << endl;
  return 0;
}
```

W projekcie `test` nie jest zdefiniowana stała preprocesora `TESTDLL_EXPORTS` więc stała `ZDLL_API` będzie mieć wartość `__declspec(dllimport)` czyli tak jak trzeba!