Wprowadzenie do tematu deklaracja przyjaźni, projekt wieloplikowy:

I. Przyjaźń:

class TKlasa2; //**deklaracja zapowiadająca**

class TKlasa1

{

public:

```
TKlasa (int pole1, in pole2);
friend class TKlasa2; //przyjaźń deklarujemy poprzez dodanie słowa friend przed

                        //nazwą klasy/funkcji, którą zaprzyjaźniamy z daną klasą
                      //oznacza to, że od teraz TKlasa2 uzyskuje dostęp do
                      //wszystkich składowych klasy TKlasa1
                      //należy pamiętać, że nie oznacza to, że TKLasa1 ma dostęp
                      //do składowych klasy TKlasa2
```

};

class TKlasa2 //definicja klasy zaprzyjaźnionej

{

};

Przyjaźń kilka pewników:

1. Jeżeli funkcja/klasa jest zaprzyjaźniona z klasą to otrzymuje dostęp do – prawa przyjaciela - do wszystkich obiektów tej klasy.
2. Funkcja zaprzyjaźniona nie jest składnikiem klasy, dlatego nie ma wskaźnika this do obiektów tej klasy.
3. Zwykle wewnątrz klasy zamieszcza się jedynie deklarację funkcji zaprzyjaźnionej.
4. Teoretycznie nie ma znaczenia, w którym miejscu klasy umieści się deklarację przyjaźni (w której etykiecie: private, public…).
5. Umownie (konwencja) jednak byłoby ją „najlepiej” umieścić w części public, bo to w niej widać składowe klasy, do których mamy dostęp spoza klasy.
6. Przyjaźń nie jest przechodnia, czyli jeżeli klasa K przyjaźni się z klasą L i M, to wcale nie oznacza, że klasy L i M przyjaźnią się ze sobą.
7. To samo tyczy się dwóch klas: jeżeli klasa K ma dostęp do obiektów klasy L, to nie znaczy, że klasa L ma dostęp do obiektów klasy K. Żeby tak było, że klasy mają wzajemny dostęp do swoich obiektów, to w każdej z nich należy zadeklarować przyjaźń z drugą klasą.

II. Projekt wieloplikowy tworzymy w oparciu o 3 pliki:

Header.h – to jest swego rodzaju spis treści, który zawiera deklaracje funkcji składowych, konstruktorów, funkcji;
zamieszczacie w nim Państwo klasę z jej składnikami, funkcjami składowymi

Przykładowy wygląd pliku nagłówkowy:

```
#include <iostream>
#include <string>

using namespace std;

class TKlasa
{

private:
	int pole1;
	int pole2;

public:
	TKlasa (int pole1, in pole2);
	friend class TZaprzyjazniona;



};


class TZaprzyjazniona
{
public:
	TZaprzyjazniona ();
	void funkcja();

private:
	TKlasa pole1;
	string skladnik1;

};
```

Do każdego pliku (header.h, source.cpp, main.cpp DÓŁĄCZAMY WYKORZYSTYWANE **W NIM** BIBLIOTEKI oraz PRZESTRZENIE NAZW

Konwencja, umowa jest taka, że w klasie najpierw umieszczamy składniki prywatne, chronione, a na końcu publiczne.

W ten sposób „na wierzchu" widać to co jest DOSTĘPNE DLA WSZYSTKICH.

Z tego samego powodu **deklarację przyjaźni warto umieścić w etykiecie publicznej.**

Source.cpp – to plik zawierający definicje – ciała funkcji składowych, konstruktorów, funkcji

Przykładowy wygląd pliku **Source.cpp:**

```
#include "Header.h"


TKlasa::TKlasa(int pole1, int pole2)
{
	this->pole1 = pole1;
	this->pole2 = pole2;
}

void TZaprzyjazniona :: funkcja()
{



}
```

DO PLIKÓW Source.cpp oraz Main.cpp DOŁĄCZACIE PAŃSTWO PLIK Header.h

Dołączając plik używamy „cudzysłowią”

Main.cpp – w tym pliku tworzycie Państwo obiekty, wywołujecie funkcje itp.

```
#include "Header.h"



int main()
{

	TZaprzyjazniona *wsk = new TZaprzyjazniona(…);
	wsk->funkcja();


	delete wsk;

	system("PAUSE");

}
```

- Tak zbudowany projekt ma na celu ułatwienie pracy grupie programistów. Jeżeli ktoś chciałby skorzystać z napisanej przez Państwa funkcji, to zajrzy do pliku source.cpp i kod będzie miał „podany na talerzu”.

- Natomiast wgląd do pliku header.h ma dać ogólny wygląd klasy, jej składników metod itp.

***JAK BUDUJEMY PROJEKT W PROGRAMIE VISUAL STUDIO:**

W ten sam sposób, jak projekty jednoplikowe, tylko tworząc „dodatkowe” pliki:

- Plik (File) -> Nowy (New) -> Projekt (Project) -> Pusty projekt (Empty Project) (nadanie nazwy projektu - Name: *własna nazwa projektu*)
- Eksplorator rozwiązań (Solution Explorer)-> Pliki źródłowe (Source Files) -> Dodaj (Add) -> Nowy element (New Item) -> Utworzenie plików projektu (source.cpp, main.cpp header.h)