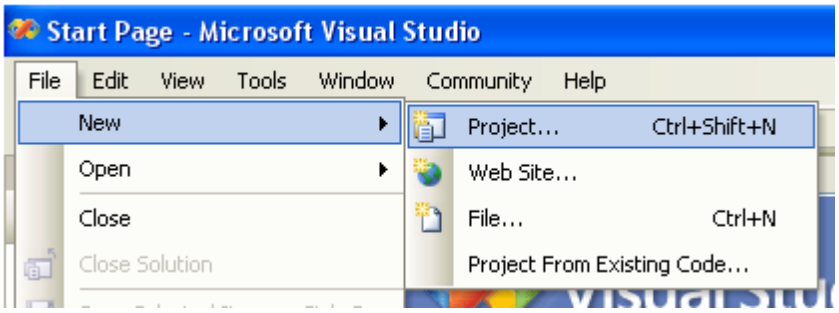
Wprowadzenie do klas w C++ oraz biblioteki openGL

1. **Bibliotek openGL.**

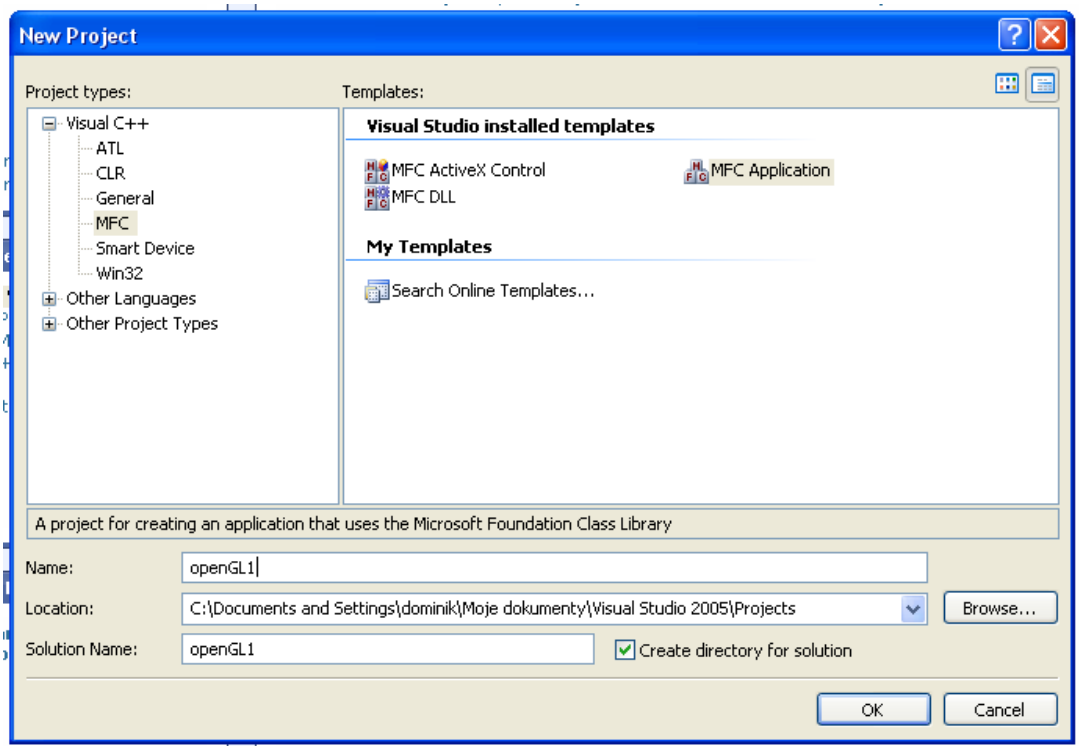
   W celu rozpoczęcia pracy z użyciem biblioteki openGL należy pobrać pliki archiwum glut-3.7.6.zip ze strony: http://www.opengl.org/resources/libraries/glut/glut_downloads.php
   Następnie rozpakować pliki:
   - glut.h do katalogu {ścieżka do Visual c++}\VC\include\gl
   - glut32.lib do katalogu {ścieżka do Visual c++}\VC\lib
   - glut32.dll do katalogu windows\system32

2. **Pierwszy projekt z wykorzystaniem biblioteki openGL**

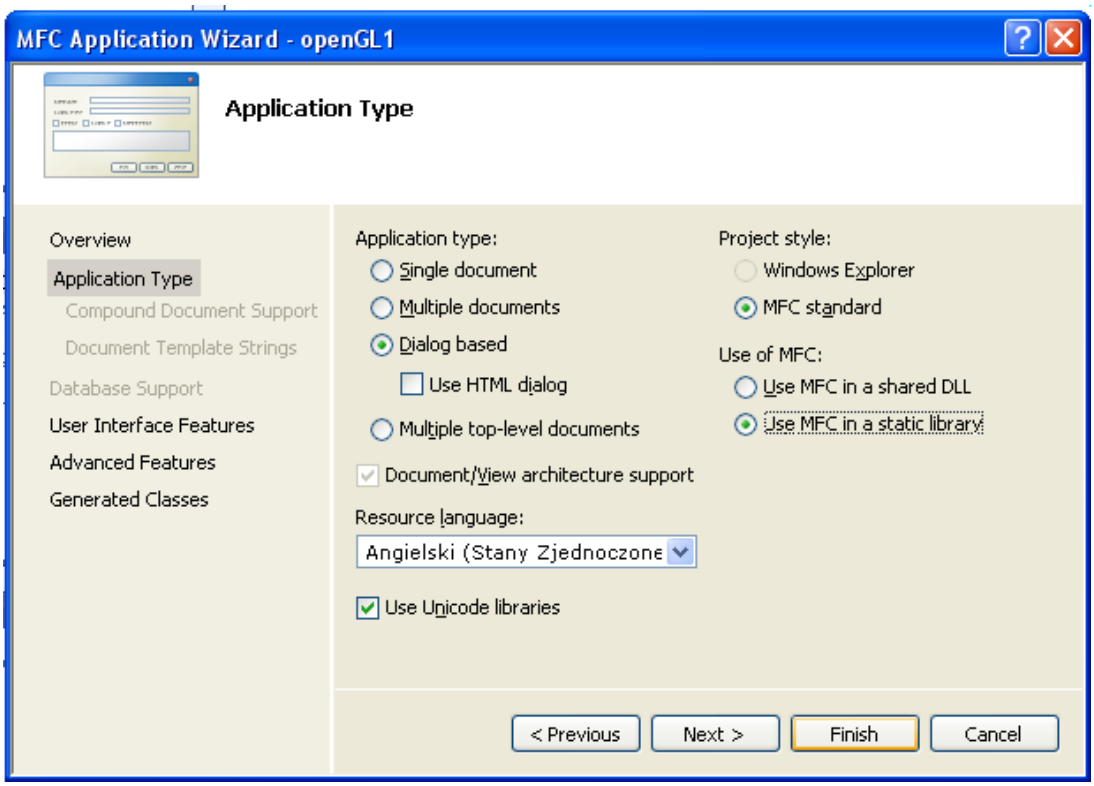
   Celem zadania jest stworzenie projektu w VS w którym możliwe będzie uruchomienie modułu openGL. Zaczynamy od stworzenia nowego projektu MFC:



   a następnie tworzymy projekt o dowolnej nazwie:



   Wybieramy „next” i ustawiamy parametry jak poniżej:



Wybieramy „Finish”. Następnie kompilujemy program i sprawdzamy czy działa. Od tego miejsca zajmować będziemy się dodawaniem klasy openGL rysującej obiekty graficzne w oknie programu. W celu dalszej pracy należy określić czym jest dodawana struktura danych.

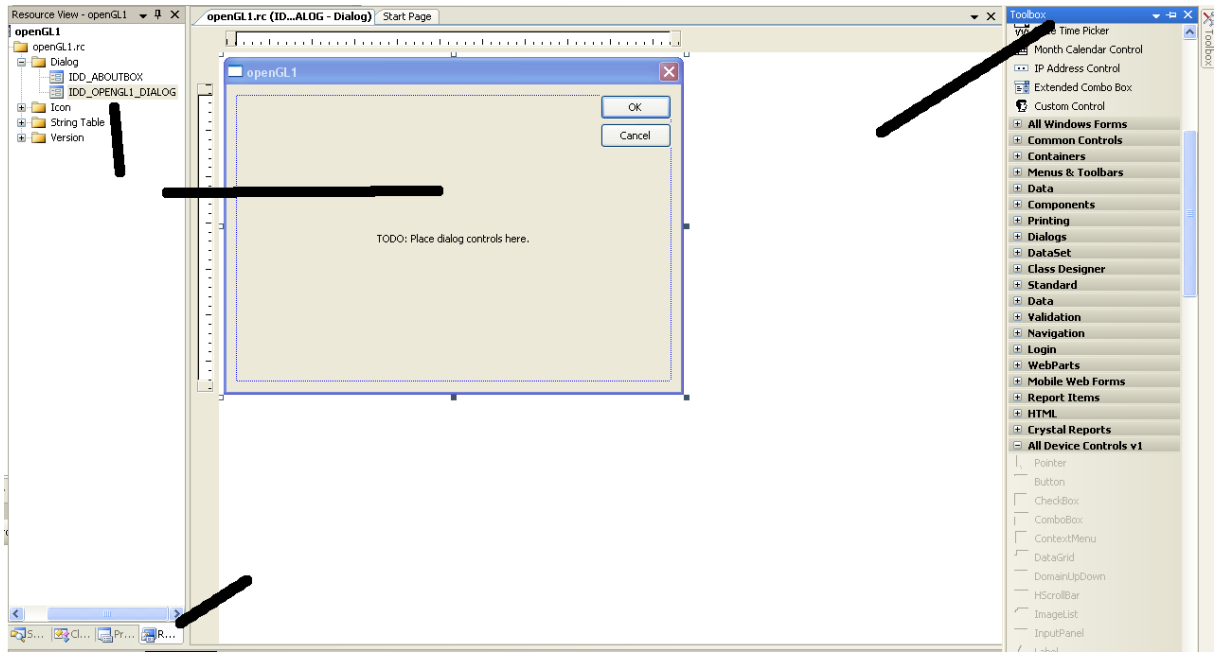
Chcemy, aby nasza „struktura danych” mogła wykonywać czynności takie jak tworzenie figur geometrycznych, wymiarowanie okna oraz wyświetlanie jego elementów. Wszelkie te funkcje można „zamknąć” w jednej formie, której nazwa określana jest jako klasa. Klasa posiada pola oraz metody. Metody są zwykłymi funkcjami natomiast pola (lub właściwości) to obiekty innych typów (np. int). W uproszczeniu klasa jest ogólnym opisem artefaktów spotykanych w środowisku (np. Opis drzewa) natomiast obiekt jest konkretnym przedmiotem (np. Konkretne drzewo rosnące w parku, o danej wysokości, odcieniu liści itp.)

Ogólnie definicja klasy:

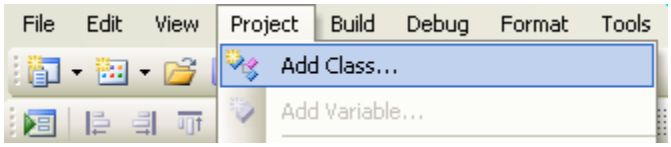
```
class nazwa_klasy
{
... // dane i metody
};
```

Klasy w c++ umożliwiają tworzenie nowych typów danych. Następnie przy ich użyciu możliwe jest tworzenie obiektów nowego typu.

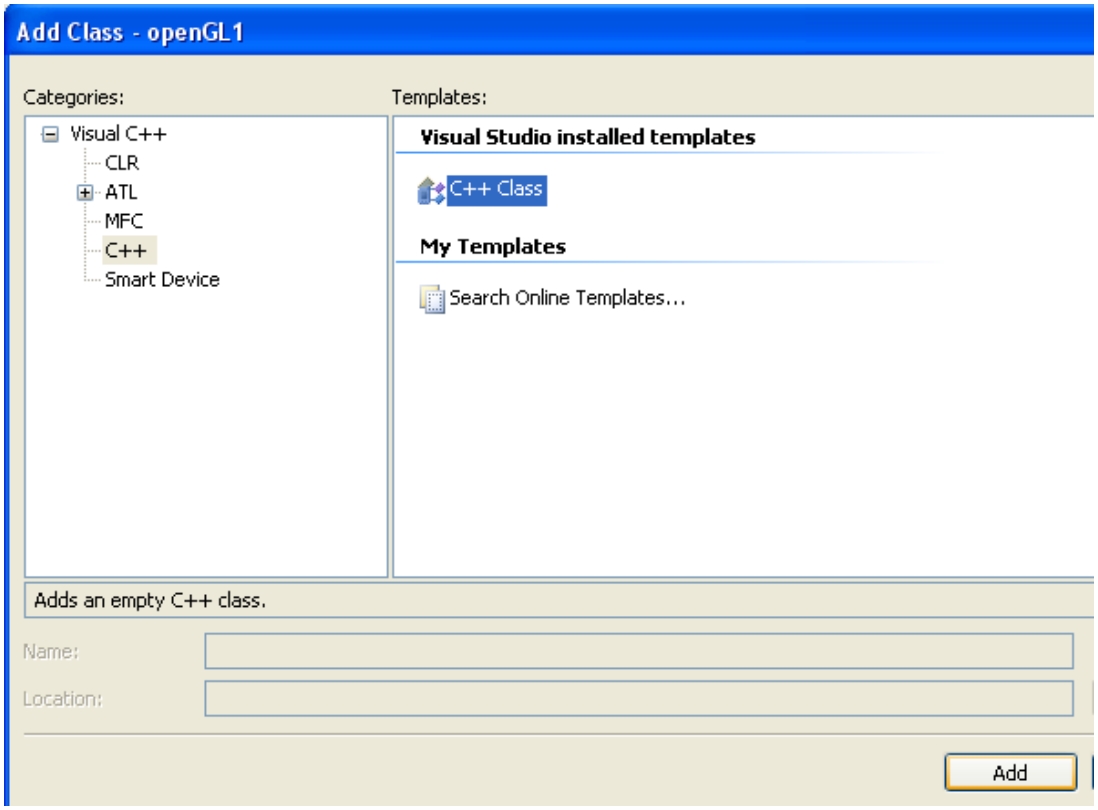
Znając już pojecie klasy możemy dodać dowolną klasę w naszym projekcie (robiliśmy to na poprzednich zajęciach jednak bez szczególnego akcentowania). Chcąc dodać naszą klasę w MFC wybieramy formularz z menu „Resources”, jak na rysunku:



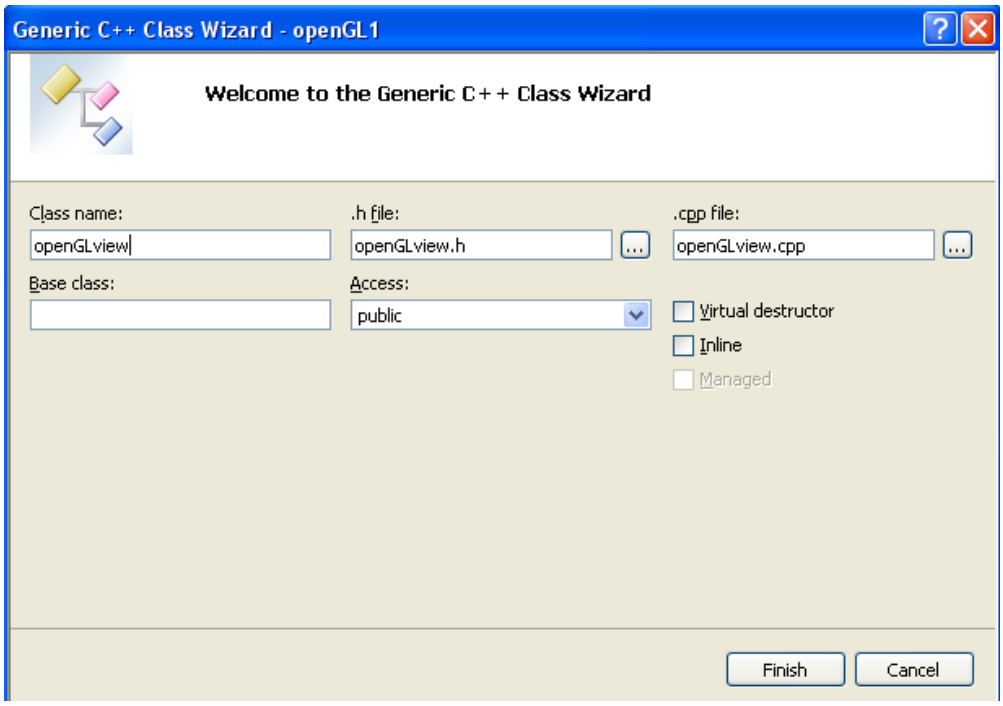
a następnie dodajemy do niego klasę (klasa będzie dodana do konkretnego dialogu wybranego jak powyżej):



Oraz wybieramy „C++ class” tak jak na rysunku:



Następnie wybieramy poniższe opcje:



Visual wygeneruje kod klasy (pliki openGLview.h i openGview.cpp). Do pliku nagłowkowego openGLview.h należy dodać linię:

```
#include <gl\glut.h> // dodajemy bibliotekę glut do projektu
```

Kompilujemy i uruchamiamy program. W przypadku problemów z kompilacją sprawdzamy czy dołączone biblioteki mają prawidłową ścieżkę.

W pliku openGLview.h zaobserwujemy deklarację klasy:

```
class openGLview
{
public:
        openGLview(void);
        ~openGLview(void);
};
```

Klasa ta zawiera dwie metody o nazwie tożsamej nazwie klasy poprzedzone etykietą *public*. Jest to postać podstawowa której metodami są konstruktor oraz destruktor klasy .

W momencie tworzenia obiektu wywoływany jest konstruktor. Jest to funkcja postaci:

nazwa_klasy(void);

Powyższa postać jest konstruktorem domyślnym, może on zostać przeciążony np.:

nazwa_klasy(nazwa_zmiennej parametr1, nazwa_zmiennej parametr2 ...);

Możliwe wtedy staje się tworzenie obiektów:

nazwa_klasy obiekt(parametr1,parametr2 ...);

co umożliwia automatyczne inicjalizowanie zmiennych wewnątrz obiektu.
Gdy obiekt jest usuwany automatycznie wywoływany jest destruktor. Jego postać to:
~nazwa_klasy(void);

W momencie kiedy usuwamy stworzony obiekt klasy dopisujemy wewnątrz destruktora kod:

```
delete obiekt1;
delete obiekt2;
….
```

Jak wcześniej zauważono nasz konstruktor oraz destruktor poprzedzone są etykietą *public* oznacza to, że taką funkcję/metodę można wywołać na zewnątrz klasy. Inne etykiety to *private* oraz *protected*. Pierwsza oznacza, że dostęp do elementu jest ograniczony tylko dla składników klasy np. metoda klasy może mieć dostęp do pola klasy (które ma etykietę private), ale już metoda z innej klasy nie. Ostatnia etykieta mówi o tym, że dostęp do elementu klasy może mieć inna klasa, która dziedziczy po danej klasie (ale o tym rozmawiać będziemy na kolejnych zajęciach).

Wracając do ćwiczenia: mam teraz chęć dodania kolejnych metod do naszej klasy *openGLview* (jak można się domyslić sam konstruktor i destruktor nie mogą wykonywać wszystkich zadań naszej klasy). Metody które należy dodać to:

```
void openGLview::display(void) {
        glClearColor(0,0,0,0);   //ustawiamy kolor tła na czarny
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); //czyścimy bufory
        glEnable(GL_DEPTH_TEST); //uruchamiamy bufor głębi
        glutSwapBuffers();//przerzuca bufor do okna używanego
        glutPostRedisplay() // oznacz  okno jako to, które będzie aktualizowane.
}
```

oraz:

```
void openGLview:: resize(int width, int height) {
        glViewport(0, 0, width, height);  //parametry okna
        glMatrixMode(GL_PROJECTION); //ustawienie kontekstu obliczeń macierzowych na stos projekcji
        glLoadIdentity(); //załadowanie macierzy jednostkowej jako aktulanej macierzy
        gluPerspective(45.0, (float)width / (float)height, 0.1, 1000.0); //ustawienie macierzy projekcji
        glMatrixMode(GL_MODELVIEW); //ustawienie kontekstu obliczeń macierzowych na stos modelu
        glLoadIdentity();  //załadowanie macierzy jednostkowej jako aktulanej macierzy
        gluLookAt( // ustawienie początkowe kamery (położenie oka, punktu ref oraz kierunku wektora )
        0.0, 0.0, 200.0,
        0.0, 0.0, 0.0,
        0.0, 1.0, 0.0);
}
```

Uwaga!: opis funkcji openGL pod adresem: http://www.opengl.org/resources/libraries/glut/spec3/spec3.html

Powyższe metody stanowią definicję zatem ich miejsce jest w pliku openGLview.cpp. Teraz modyfikacji należy poddać również plik openGLview.h, gdzie dopisane zostają deklaracje metod. Wygląd klasy powinien być następujący:

```
class openGLview
{
        public:
        openGLview(void);
        public:
        ~openGLview(void);
        void display(void);
        void resize(int width, int height);
};
```

Mając już metody można wywołać ich działanie. Dokonujmy tego po naciśnięciu przycisku „Ok” w oknie dialogowym. Aby tego dokonać należy wejść na zakładkę „”Resources” i w oknie kliknąć dwa razy „Ok”. Funkcja wykonująca się po naciśnięciu tego przycisku powinna wyglądać następująco:

```
void CopenGL1Dlg::OnBnClickedOk()
{
	// TODO: Add your control notification handler code here
	glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);//ustawienie kolorów, precyzji i głębi
	glutCreateWindow("openGL test");//tworzymy nowe okno
	// ustawienie odpowiednich funkcji, które będą wywoływane przy odpowiednich zdarzeniach:
	glutDisplayFunc(display);//ustawiamy funkcje display
	glutReshapeFunc(resize);//ustawiamy funkcje resize
	this->ShowWindow(SW_HIDE);// ukrywamy główne okno
	glutMainLoop(); //uruchamiamy pętlę openGL
}
```

Na początku pliku openGL1Dlg.cpp musimy również dodać deklarację obiektu openGLview i odpowiednie definicje funkcji, tak aby kod wyglądał następująco:

```
#include "stdafx.h"
#include "openGL1.h"
#include "openGL1Dlg.h"
#include "openGLview.h"
#include <gl\glut.h> // glut.h includes gl.h and glu.h

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

openGLview gl;  // deklaracja

void display(void)
{
gl.display();
}
void resize(int w, int h)
{
gl.resize(w, h);
}

…
```

Kompilujemy i uruchamiamy program – pojawi się okno openGL.
Następnie aby narysować figury na scenie tworzymy funkcję w pliku openGLview.cpp:

```
void openGLview::rysuj_figury(void) {
	//punkt
	glColor3f(1,0,0);//czerwony kolor
	glBegin(GL_POINTS);//rysujemy punkt
	glVertex3f(0.0, 0.0, 0.0);
	glEnd();
	//linia
	glColor3f(0,1,0);//zielony kolor
	glBegin(GL_LINES); //linia
	glVertex3f(0.0, 0.0, 0.0);
	glVertex3f(110.0, 110.5, 0.0);
	glEnd();
```

```
        //trójkąt
        glColor3f(0,0,1);//niebieski
        glBegin(GL_TRIANGLES);//trojkat
        glVertex2f(30,-40);
        glVertex2f(35,-30);
        glVertex2f(40,-40);
        glEnd();
        //kwadrat
        glColor4f( 1.0, 1.0, 0.0, 1.0 );
        glBegin( GL_QUADS ); //kwadrat
        glVertex2f( -20.0f, -20.0f );
        glVertex2f( -20.0f, 20.0f );
        glVertex2f( 20.0f, 20.0f );
        glVertex2f( 20.0f, -20.0f );
        glEnd();
// lub rysujemy bezpooerednio całe figury skladające sie z krawędzi:
//glColor3f(1.0f, 0.0f, 0.0f); //ustawiamy kolor
//glutWireCube(30); //szescian
//glutWireSphere(50.0f, 30, 30); //sfera
//glutWireCone(50.0f, 50.0f, 30, 30); // i inne - do sprawdzenia w dokumentacji:
//http://www.opengl.org/resources/libraries/glut/spec3/spec3.html
}
```

Musimy również dodać deklaracje metody w pliku openGLview.h:

```
void rysuj_figury(void);
```

a następnie wywołanie w funkcji display() w odpowiednim miejscu:

```
….
glEnable(GL_DEPTH_TEST); //uruchamiamy bufor g3ebi
rysuj_figury(); //testowe rysowanie figur
glutSwapBuffers();
glutPostRedisplay(); // marks the current window as needing to be redisplayed

…
```

Program pozwolił na wyrysowanie figur w oknie. Jednak najważniejszą częścią ćwiczenie jest tworzenie obiektów naszej własnej klasy („openGLview gl”). Możliwe jest też tworzenie wskaźników do obiektów: „openGLview* gl” jednak wtedy odnoszenie się do składników klasy odbywa się poprzez operator strzałki „->”, a nie poprzez operator kropki. Przykład

-(dla wskaźnika) gl->rysuj_figury();

- (dla obiektu) gl.rysuj_figury();

Obok deklaracji obiektów istotne jest tworzenie definicji metod: dobrym nawykiem jest dodawanie definicji metod w pliku .cpp klasy. Można również dodać wewnątrz klasy, jednak jest to uzasadnione/możliwe tylko przy pewnych warunkach.

Sprawozdanie:

W sprawozdaniu należy odpowiedzieć na pytanie:

1. Czy zawsze powinno stosować się etykiety *public* ?

2. Kiedy definicję metody można wpisać wewnątrz klasy?
3. Czy etykiety *public*, *private* oraz *protected* można stosować kilkukrotnie w obrębie ciała klasy?
4. Co się stanie jeśli klas nie będzie miała żadnej etykiety?
5. Jaki jest cel stosowania konstruktorów klasy?
6. Napisz kod klasy dla której wykonano dwukrotne przeciążenie konstruktora (dwa różne konstruktory). Uzasadnij użycie przeciążenia konstruktora (krótki przykład, max dwie metody oraz max trzy pola)