KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Instrukcja współfinansowana przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego
w projekcie

*„Innowacyjna dydaktyka bez ograniczeń
– zintegrowany rozwój Politechniki Łódzkiej – zarządzanie Uczelnią,
nowoczesna oferta edukacyjna i wzmacniania zdolności
do zatrudniania osób niepełnosprawnych”*

Instrukcja jest dystrybuowana bezpłatnie.

Instrukcja do laboratorium, część 2.

Piotr M. Szczypiński

Programy komputerowe do przetwarzania i analizy obrazów oraz wideo

Zestawienie praktycznych informacji dotyczących programowania na potrzeby przetwarzania obrazów

Zadanie nr 13 – Studia podyplomowe „Przetwarzanie i analiza obrazów biomedycznych”

Politechnika Łódzka
Instytut Elektroniki

90-924 Łódź, ul. Żeromskiego 116,
tel. 042 631 28 83
www.kapitalludzki.p.lodz.pl

Wstęp – dlaczego programowanie

Umiejętność programowania w językach kompilowanych jest przydatna a często nieodzowna do rozwiązania nietypowych problemów związanych z przetwarzaniem obrazów. Gotowe programy do edycji i przetwarzania obrazów, takie jak *GIMP* lub *ImageJ*, opracowano tak, aby umożliwiały wykonywanie typowych operacji na obrazach. Umożliwiają one przeprowadzenie podstawowych filtracji, korekcji barwy obrazu oraz segmentację. Wymagają one ponadto dużego zaangażowania osoby obsługującej taki program i praktycznie nie dają możliwości pełnej automatyzacji procesu przetwarzania w przypadku dużej liczby obrazów.

Do rozwiązania problemów nietypowych, uruchomienia sekwencji metod lub przygotowania algorytmu „szytego na miarę" konieczne jest opracowanie własnego programu komputerowego. Program taki można utworzyć wykorzystując bądź to interpretowane języki programowania (tekst programu jest analizowany podczas jego wykonywania), bądź kompilowane (tekst programu jest tłumaczony na kod maszynowy przed jego wykonaniem).

W przypadku języków kompilowanych, tekst programu tłumaczony jest na kod maszynowy programu (zrozumiały dla procesora, bezpośrednio sterujący pracą procesora). Tworzony jest program wynikowy – wykonywalny. Podczas wykonywania takiego programu, procesor nie traci czasu na analizę tekstu źródłowego. Przewagą języków kompilowanych nad interpretowanymi jest zatem większa szybkość działania. Przykładami kompilowanych języków programowania są języki **C** i **C++**. W ramach zajęć, do tworzenia programów w językach C i C++ wykorzystane zostanie środowisko programistyczne Microsoft Visual Studio C++.

Tworząc program komputerowy wykorzystujemy zazwyczaj tzw. biblioteki funkcji (procedur). Biblioteki funkcji są modułami, które dołącza się do własnoręcznie pisanych programów. Biblioteki zawierają przydatne procedury napisane przez innych programistów. Z procedur zawartych w bibliotekach można korzystać we własnym programie. Nie ma wówczas potrzeby pisania własnej procedury o ile taka procedura została już prze kogoś napisana i udostępniona w formie biblioteki. Korzystanie z bibliotek znacznie usprawnia i ułatwia pracę programistyczną. Jedną z bibliotek przydatną podczas programowania na potrzeby przetwarzania obrazów jest **OpenCV**.

Cel

Celem jest zdobycie podstawowych umiejętności dotyczące programowania w języku C i C++. Umiejętności te powinny umożliwić pisanie własnych programów do przetwarzania dwuwymiarowych, rastrowych obrazów cyfrowych.

Cele szczegółowe:

1. Zrozumienie różnicy między językami interpretowanymi a kompilowanymi.
2. Zaznajomienie się z prostymi programami w C i C++.
3. Umiejętność utworzenie projektu programu w środowisku programistycznym.
4. Zrozumienie celu kompilacji, konsolidacji i debuggowania.
5. Zrozumienie pojęcia biblioteki funkcji.

6. Umiejętność stworzenia prostej biblioteki funkcji oraz wykorzystującego ją programu.

Podstawowe pojęcia

Poniżej znajdują się definicja cytowane za http://pl.wikipedia.org/

Aby program napisany w danym języku mógł być wykonany, niezbędne jest odpowiednie przetworzenie jego kodu źródłowego:

- Kompilacja – kod źródłowy jest tłumaczony do postaci kodu maszynowego, czyli sekwencji elementarnych operacji gotowych do bezpośredniego przetworzenia przez procesor komputera. Jeżeli dany język programowania podlega kompilacji, określany jest mianem *kompilowanego języka programowania*.
- Interpretacja – kod źródłowy jest na bieżąco tłumaczony i wykonywany przez dodatkowy program zwany interpreterem. Jeżeli język podlega interpretacji, nazywany jest *interpretowanym językiem programowania*.

Kompilacja to proces automatycznego tłumaczenia kodu napisanego w języku programowania na kod maszynowy. Dane wejściowe najczęściej nazywa się kodem źródłowym. Program wykonujący tłumaczenie to kompilator. Przeważnie kompilacja jest częścią większego procesu tłumaczenia, tworzony w jej trakcie kod wynikowy jest przekazywany do innych programów (np. linkera), możliwe jest również tłumaczenie do postaci zrozumiałej dla człowieka.

Kod maszynowy to postać programu komputerowego (zwana postacią *wykonywalną* lub *binarną*) przeznaczona do bezpośredniego lub prawie bezpośredniego wykonania przez procesor. Jest ona dopasowana do konkretnego typu procesora i wyrażona w postaci rozumianych przez niego kodów rozkazów i ich argumentów. Jest to postać trudna do analizy przez człowieka.

Konsolidacja (**linkowanie** od ang. *link* - *łączyć*) to proces polegający na połączeniu skompilowanych modułów (plików zawierających kod obiektowy lub plików bibliotek statycznych) i utworzeniu pliku wykonywalnego lub rzadziej innego pliku obiektowego. Dodatkowo podczas konsolidacji do pliku wynikowego mogą być dołączone odpowiednie nagłówki i informacje charakterystyczne dla konkretnego formatu pliku wykonywalnego.

Debug tool, **Debugger** (czytaj *debager* - z ang. odpluskwiacz) – program komputerowy służący do dynamicznej analizy innych programów, w celu odnalezienia i identyfikacji zawartych w nich błędów, zwanych z angielskiego **bugami** (robakami). Proces nadzorowania wykonania programu za pomocą debuggera określa się mianem debugowania.

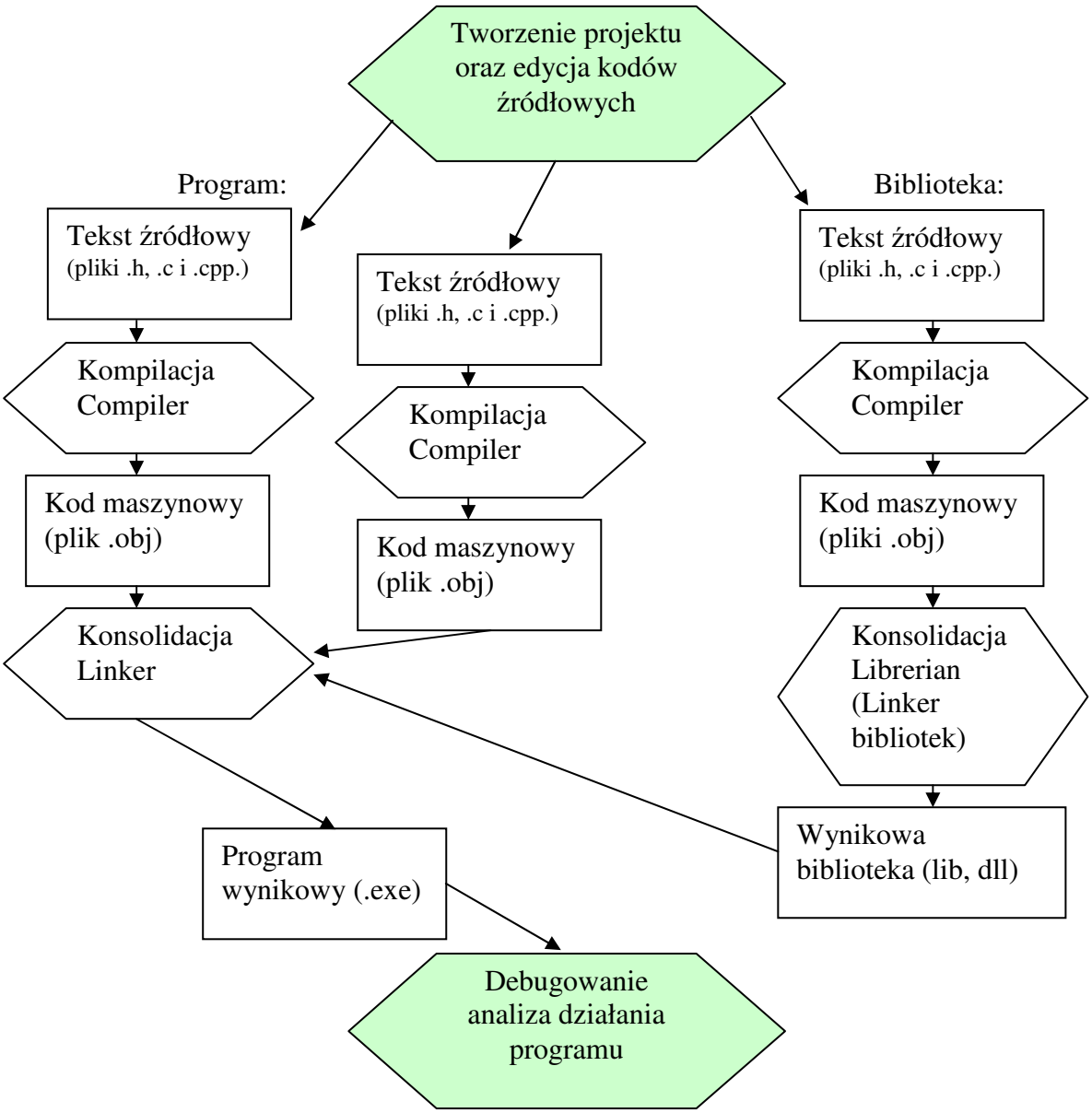
Podstawowym zadaniem debuggera jest sprawowanie kontroli nad wykonaniem kodu, co umożliwia zlokalizowanie instrukcji odpowiedzialnych za wadliwe działanie programu. Współczesne debuggery pozwalają na efektywne śledzenie wartości poszczególnych zmiennych, wykonywanie instrukcji krok po kroku czy wstrzymywanie działania programu w określonych miejscach. Debugger jest standardowym wyposażeniem większości współczesnych środowisk programistycznych.

Narzędzia: edytor tekstu źródłowego, kompilator, konsolidator i debugger stanowią elementy składowe zintegrowanego środowiska programistycznego

Zintegrowane środowisko programistyczne (ang. *Integrated Development Environment*, IDE) jest to aplikacja lub zespół aplikacji (środowisko) służących do tworzenia, modyfikowania, testowania i konserwacji oprogramowania.

Aplikacje będące zintegrowanymi środowiskami programistycznymi charakteryzują się tym, że udostępniają złożoną, wieloraką funkcjonalność obejmującą edycję kodu źródłowego, kompilowanie kodu źródłowego, tworzenie zasobów programu (tzn. formatek / ekranów / okien dialogowych, menu, raportów, elementów graficznych takich jak ikony, obrazy itp.), tworzenie baz danych, komponentów i innych.

Etapy tworzenia programu



Tworzenie programu komputerowego polega na utworzeniu projektu programu, jego kompilacji, konsolidacji, analizie jego działania oraz wprowadzania ew. korekt. Tworzenie projektu polega na określeniu liczby i rodzaju plików źródłowych, wykorzystywanych bibliotek, konfiguracji sposobu kompilacji i konsolidacji. Kolejnym etapem jest napisanie

tekstów źródłowych naszego programu bądź modułu programu (np. biblioteki). Po napisaniu programu program jest kompilowany i konsolidowany. Jest to proces automatycznie przeprowadzany przez środowisko programistyczne. Podczas tego procesu środowisko sprawdza czy program został napisany poprawnie: jeśli nie – informuje o błędach, jeśli tak – tworzy końcowy program wykonywalny. Kolejnym etapem jest analiza sposobu działania programu (debugowanie), czy program działa zgodnie z oczekiwaniami.

Tworząc projekt należy podać informacje potrzebne kompilatorowi i konsolidatorowi (linkerowi). Niewłaściwe informacje lub ich brak może uniemożliwić wytworzenie programu wynikowego lub spowodować, że program wynikowy nie będzie działał zgodnie z oczekiwaniami.

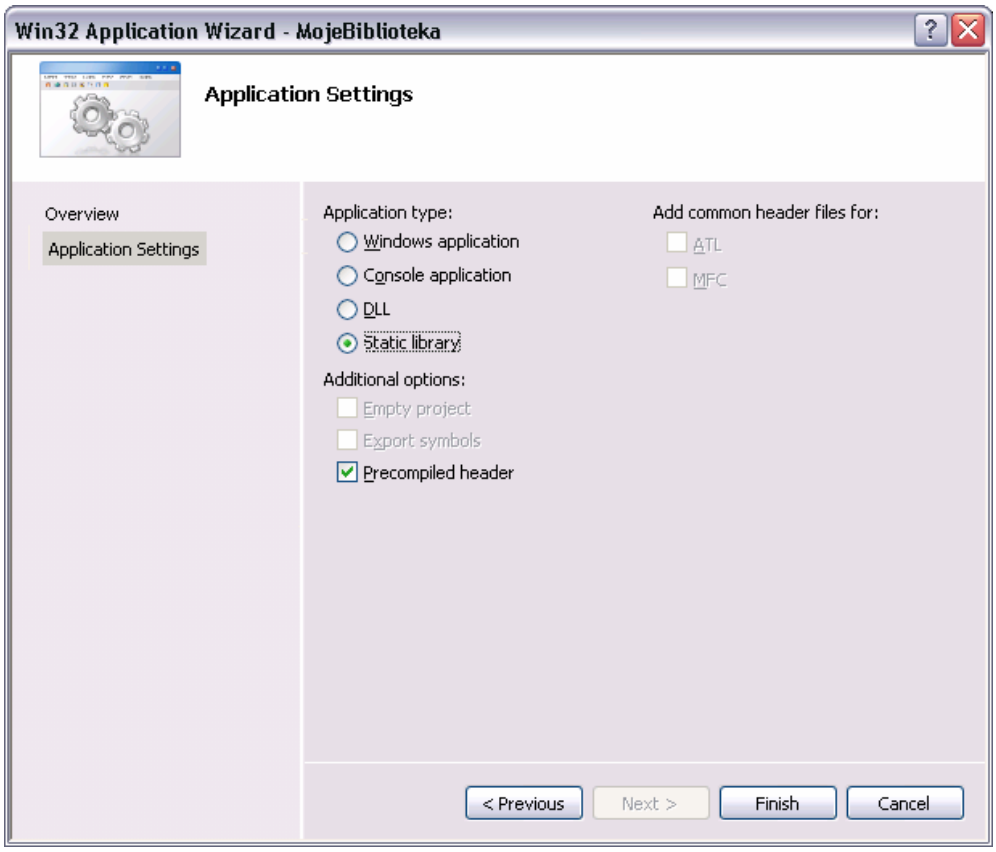
Co musi „wiedzieć" kompilator:

1. Gdzie znajdują się pliki do skompilowania i pliki z deklaracjami (opisem) funkcji bibliotecznych (podajemy nazwy katalogów, tzw. ścieżki dostępu).
2. Które pliki przeznaczone są do skompilowania (pliki widoczne są jako elementy składowe projektu)

Co musi „wiedzieć" linker:

3. Gdzie znajdują się pliki bibliotek dołączanych do programu (podajemy nazwy katalogów, tzw. ścieżki dostępu).
4. Musi znać listę plików obiektowych (z kodem maszynowym) i bibliotek, które ma ze sobą połączyć.

Decyzję o tym czy celem projektu będzie tworzenie biblioteki czy programu dokonuje się podczas tworzenia nowego projektu. W środowisku Visual Studio w celu utworzenia nowego projektu należy wybrać opcję: *File-> New->Project.* Wykorzystując kreator projektu w kolejnych okienkach podajemy informacje dotyczące tego, jaki projekt chcemy utworzyć. Do wyboru mamy projekt programu z interfejsem graficznym (*Windows application*), program konsolowy, działający w trybie tekstowym (*Console application*), oraz biblioteki dołączane statycznie (*Static library*) lub dynamicznie (*DLL*).



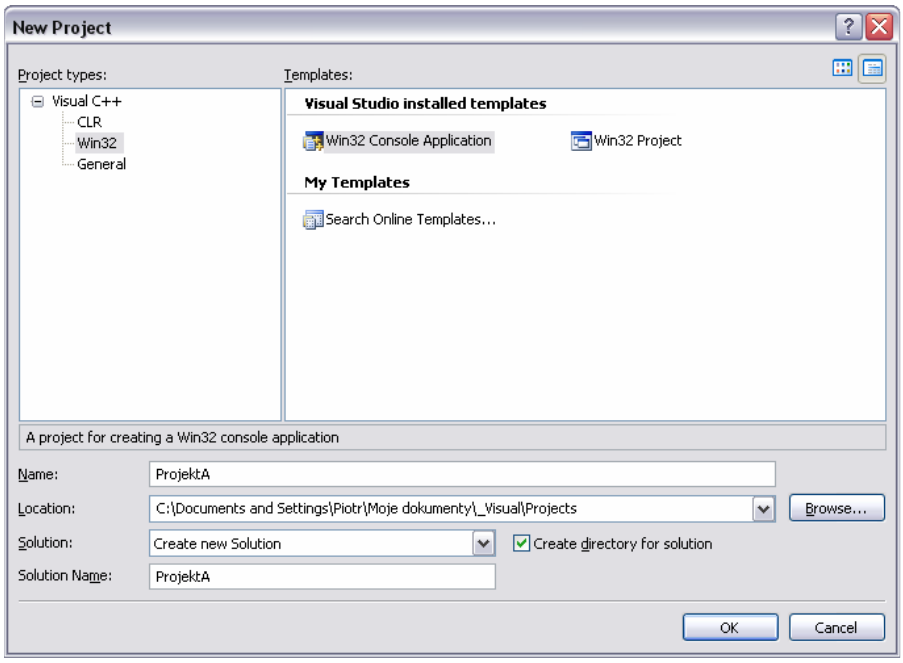
Tworzenie projektu programu wykorzystującego OpenCV

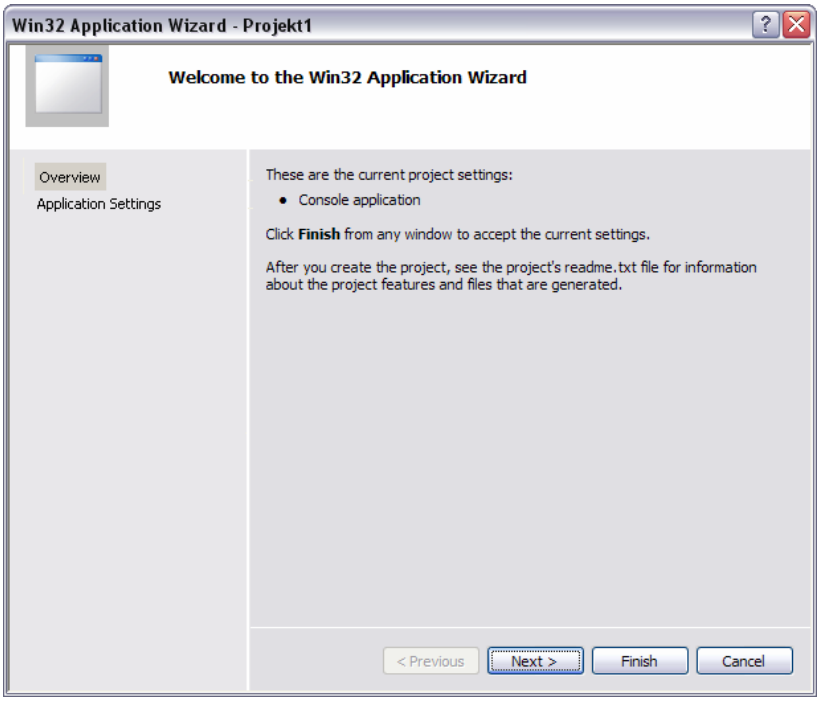
Utworzenie nowego projektu

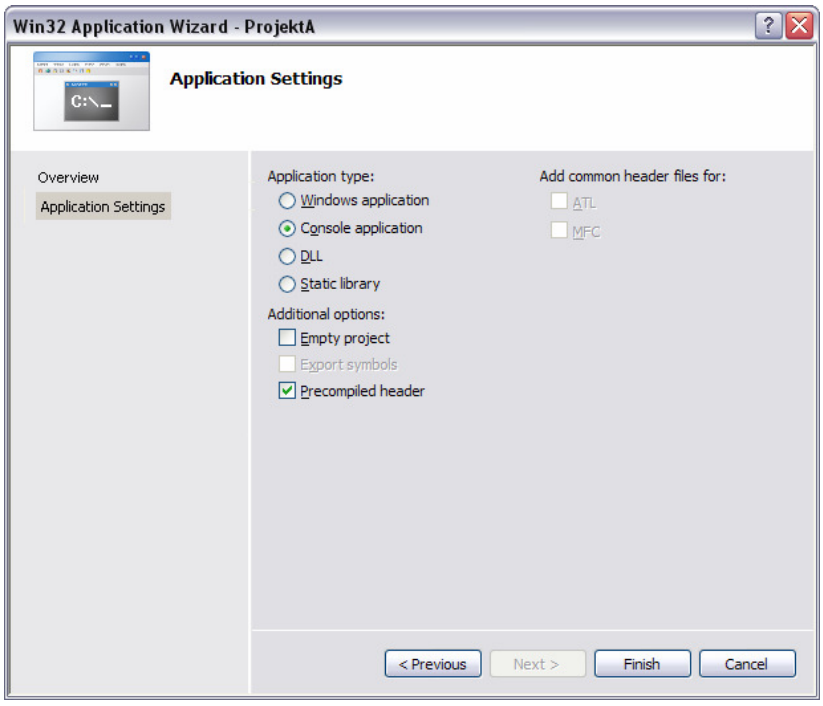
1. Uruchomić Visual Studio C++ Express Edition

2. Należy wybrać: *File-> New->Project*

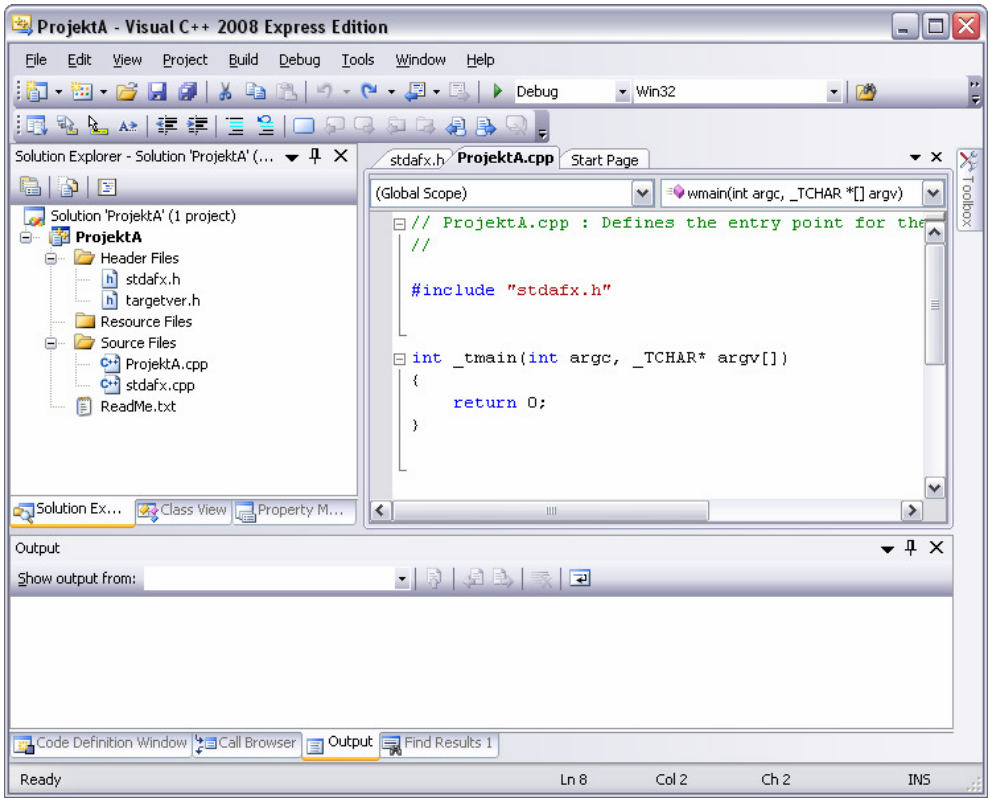
3. Korzystając z kreatora projektów wybrać rodzaj projektu dla systemu 32-bitowego (Win32), aplikacja konsolowa (Console application), podać nazwę projektu i katalog, w którym ma zostać utworzony.


New Project
Project types:
Templates:
Visual C++
CLR
Win32
General
Visual Studio installed templates
Win32 Console Application
Win32 Project
My Templates
Search Online Templates...
A project for creating a Win32 console application
Name:
ProjektA
Location:
C:\Documents and Settings\Piotr\Moje dokumenty_Visual\Projects
Browse...
Solution:
Create new Solution
Create directory for solution
Solution Name:
ProjektA
OK
Cancel


Win32 Application Wizard - Projekt1
Welcome to the Win32 Application Wizard
Overview
Application Settings
These are the current project settings:
Console application
Click Finish from any window to accept the current settings.
After you create the project, see the project's readme.txt file for information about the project features and files that are generated.
< Previous
Next >
Finish
Cancel

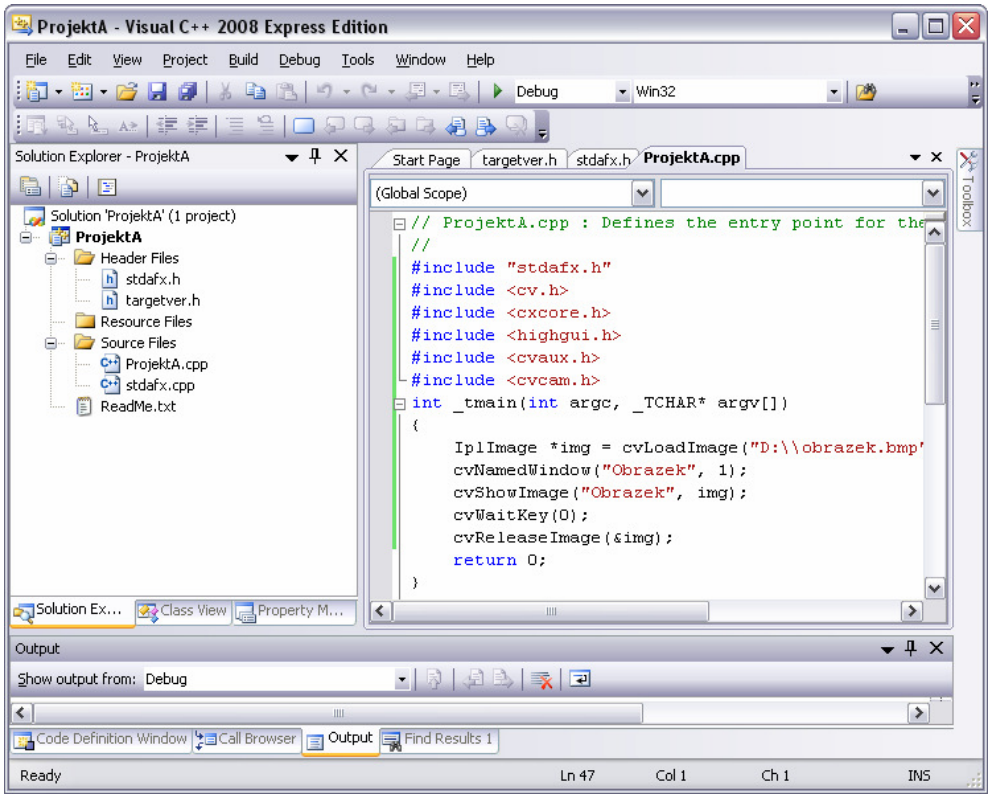


4. Po utworzeniu projektu pojawia się okno środowiska programistycznego



Edycja kodu źródłowego

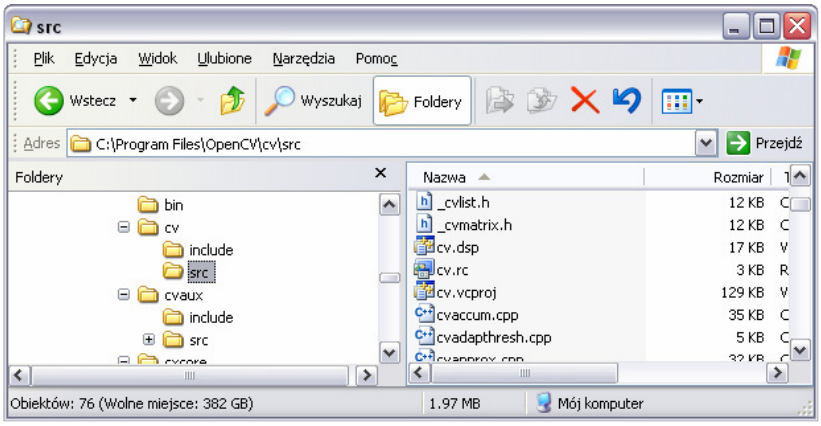
5. Należy dokonać edycji pliku źródłowego projektu (plik z rozszerzeniem *.cpp*). Należy pamiętać o dodaniu plików nagłówkowych bibliotek. (Przykładowy tekst źródłowy korzysta z biblioteki OpenCV i zawiera odpowiednie pliki nagłówkowe).

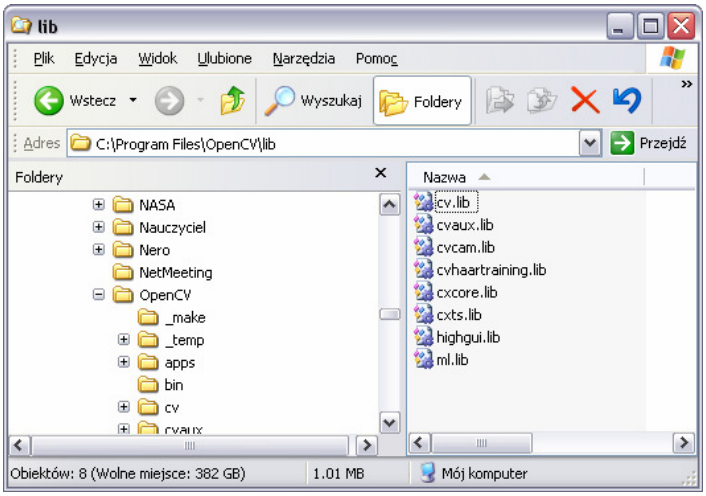


Ustawienie parametrów kompilacji i konsolidacji

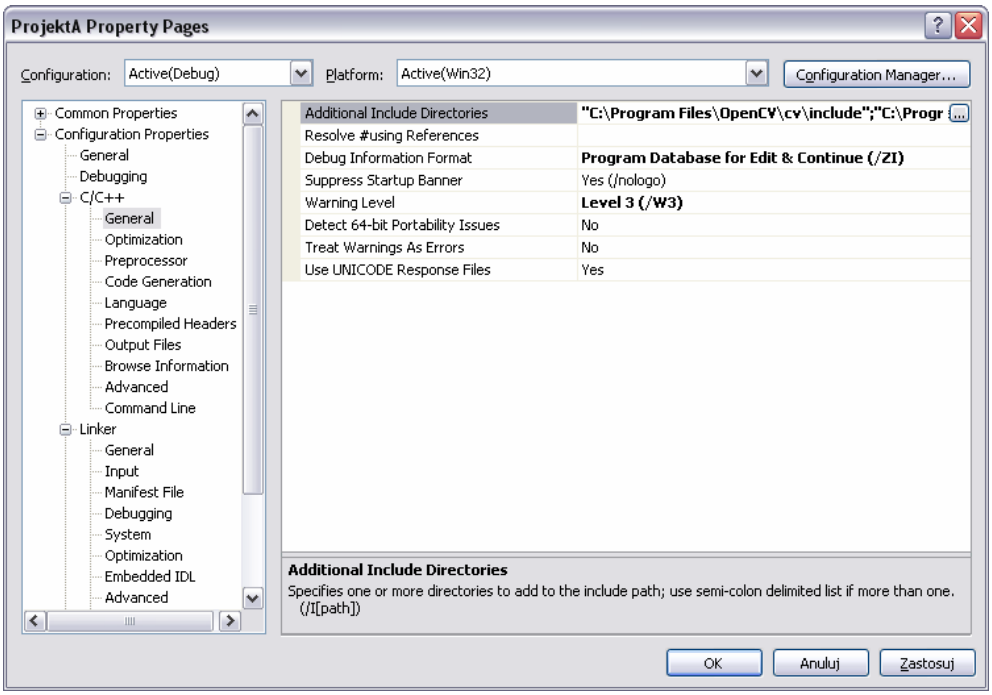
6. Klikamy prawym guzikiem myszy na wytłuszczonej nazwie projektu w panelu Solution Explorer i wybieramy właściwości (Properties).

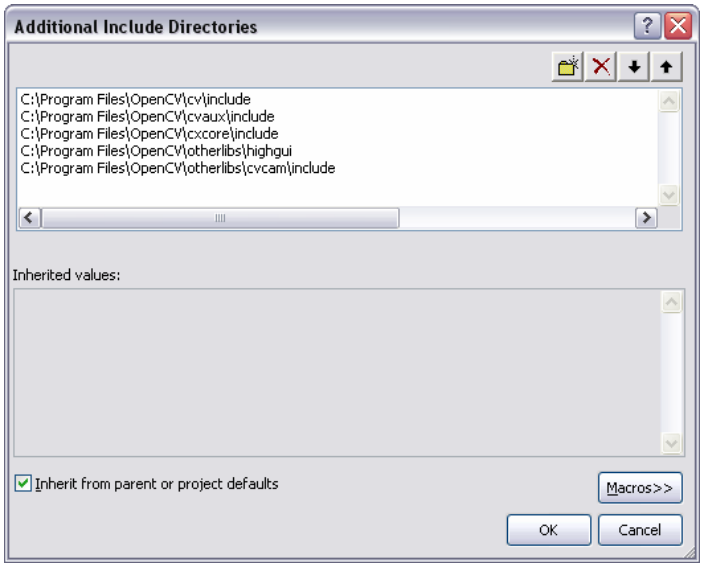
7. Upewniamy się czy na komputerze zainstalowane są biblioteki OpenCV. Sprawdzamy gdzie znajdują się pliki nagłówkowe (.h) i pliki bibliotek (.lib).



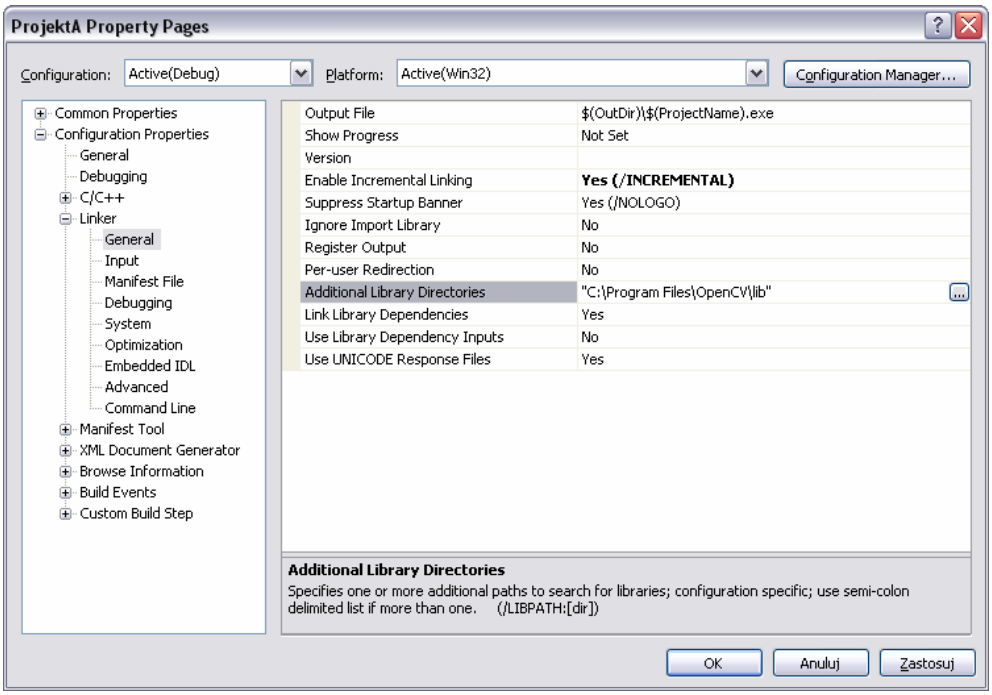


8. Wybieramy opcje kompilatora (C/C++) i uzupełniamy opcję *Additional Include Directories* podając nazwy folderów zawierających pliki nagłówkowe biblioteki OpenCV.

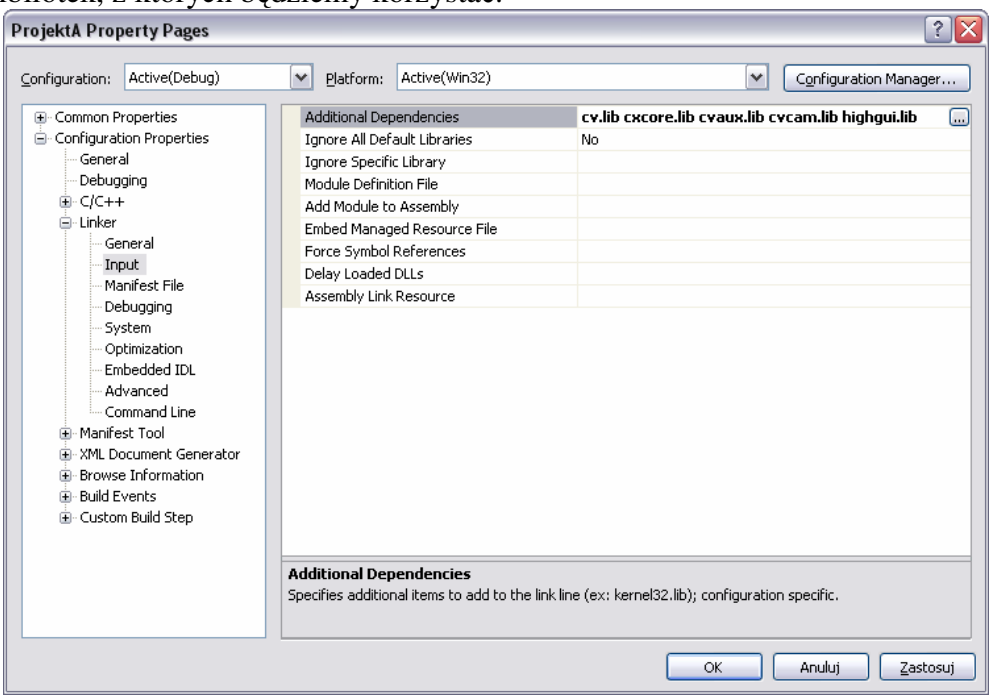




9. Wybieramy opcje Linker->General i w polu *Additional Library Directories* dodajemy ścieżki dostępu do bibliotek (przykład biblioteki OpenCV).



10. Wybieramy panel opcji *Linker -> Input* i w polu *Additional Dependencie*s podajemy nazwy bibliotek, z których będziemy korzystać.



11. Wciskamy Zastosuj i OK.

Kompilacja, konsolidacja i uruchomienie

12. Dokonujemy kompilacji naszego programu, jego konsolidacji z bibliotekami oraz uruchomienia. Środowisko programistyczne wykona wszystkie te etapy automatycznie, jeśli wciśniemy guzik *Run* (zielony trójkącik).

13. Jeśli kod źródłowy lub projekt zawiera błędy wówczas program nie zostanie uruchomiony. W tym przypadku w panelu *Output* środowiska pojawi się lista błędów wykrytych w fazie kompilacji lub konsolidacji.

W linijce opisu błędu kompilacji występuje nazwa pliku, w którym wystąpił błąd, w nawiasie numer kolejny linijki w tym pliku gdzie błąd występuje, słowo *error* (ang. błąd), numer błędu oraz syntetyczny opis rodzaju błędu w języku angielskim. Przykład informacji o błędzie kompilacji:

```
  1>Compiling...
...
  1>c:\...\projekta.cpp(11) : error C2660: 'cvLoadImage' : function does not take 3 arguments
```

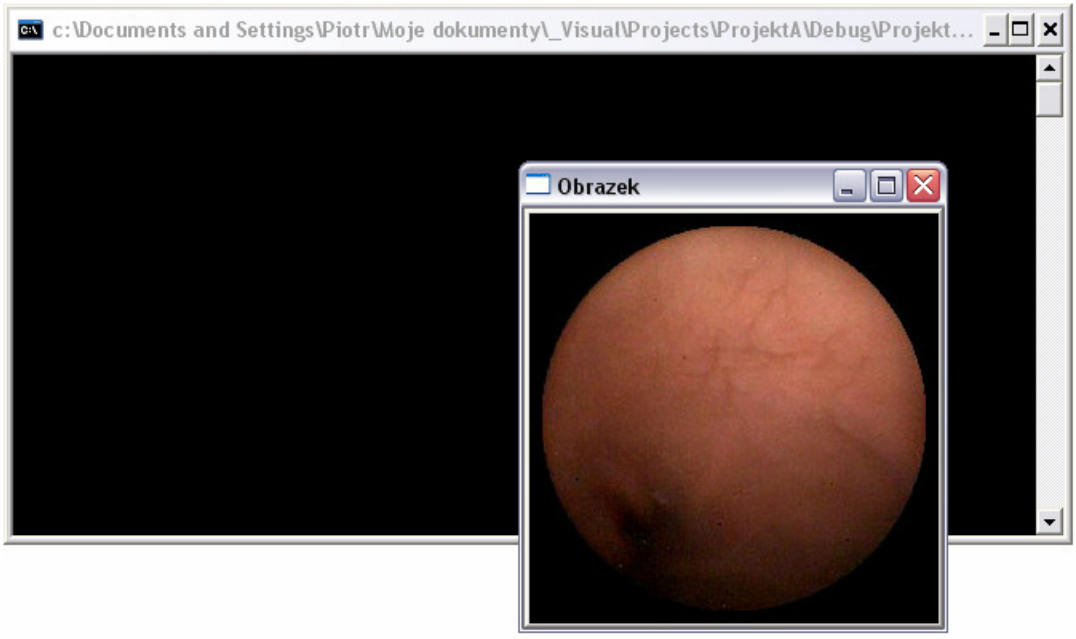
Powiższy błąd wynika z nieprawidłowej liczby argumentów przekazywanych do funkcji *cvLoadImage*.

W linijce opisu błędu konsolidacji występują słowa *LINK* i *fatal error*, numer/kod błędu oraz syntetyczny opis rodzaju błędu w języku angielskim. Przykład informacji o błędzie konsolidacji:

```
1>Linking...
...
1>LINK : fatal error LNK1104: cannot open file 'cxcore.lib'
```

Powiższy błąd wynika z tego, że nie podano katalogu zawierającego biblioteki w polu *Additional Library Directories* opcji linkera.

14. Efektem prawidłowo działającego programu jest wyświetlenie okienka konsolowego oraz okienka z obrazkiem.



Kod źródłowy

Struktura przykładowego programu jest bardzo prosta. Nie zawiera on dodatkowych funkcji poza bibliotecznymi, nie zawiera też pętli powodujących wielokrotne wykonanie fragmentów kodu. Kolejność wywoływania i wykonywania poszczególny instrukcji lub funkcji jest więc zgodna z ich kolejnością w tekście kodu źródłowego.

Pierwszych kilka linijek kodu odpowiedzialnych jest za dołączenie do tekstu programu tzw. plików nagłówkowych bibliotek. W plikach nagłówkowych podane są informacje dotyczące nazw funkcji dostępnych w bibliotece oraz liczby i rodzaju argumentów przyjmowanych przez te funkcje. To dzięki opisowi (deklaracjom) funkcji zawartym w plikach nagłówkowych kompilator „wie" w jaki sposób powinniśmy wywoływać funkcje w naszym programie.

```
#include "stdafx.h"
#include <cv.h>
#include <cxcore.h>
#include <highgui.h>
#include <cvaux.h>
#include <cvcam.h>
```

Dla przykładu otwórzmy plik nagłówkowy *highgui.h*. W pliku tym znajdziemy między innymi deklarację funkcji cvLoadImage, która wygląda następująco

```
CVAPI(IplImage*) cvLoadImage(const char* filename, int iscolor
CV_DEFAULT(CV_LOAD_IMAGE_COLOR));
```

Jest to informacja dla kompilatora, że funkcja zwraca wskaźnik do obrazu w formacie *IplImage*, ma nazwę *cvLoadImage* i przyjmuje dwa argumenty, pierwszy jest nazwą pliku (*filename*) a drugi wartością całkowitą *int* o nazwie *iscolor*.

Program rozpoczyna działanie wywołanie funkcji *main*, *_tmain* lub *WinMain*. W naszym przykładzie funkcja ta ma nazwę *_tmain*. Funkcja ta przyjmuje dwa argumenty. Ciało funkcji, czyli instrukcje wykonywane przez program po wywołaniu tej funkcji, znajdują się pomiędzy nawiasami {}.

```
int _tmain(int argc, _TCHAR* argv[])
{
...
}
```

Wewnątrz ciała funkcji umieszczone są w kolejnych linijkach definicje zmiennych oraz nazwy funkcji, które maja zostać wywołane. Poszczególne linijki kończą znaki średnika.

W pierwszej linijce funkcji *_tmain* deklarowany jest wskaźnik do obrazu typu *IplImage* o nazwie *img*. O tym, że ma to być wskaźnik decyduje symbol gwiazdki *. Wskaźnik określa miejsce w pamięci komputera, w którym znajduje się interesujący nas rodzaj danych (tutaj jest to obraz). Używanie wskaźników pozwala przyspieszyć działanie programu, np. zamiast kopiować duży obraz w celu przekazania go innej funkcji, prościej jest przekazać tej funkcji informację (wskaźnik), w którym miejscu pamięci ten obraz się znajduje.

```
IplImage *img = cvLoadImage("D:\\obrazek.bmp", 1);
```

Funkcja *cvLoadImage* zajmuje (alokuje) w pamięci komputera odpowiednio dużo miejsca na wpisanie tam obrazka. Następnie odczytuje obraz z pliku o podanej nazwie i kopiuje go do zaalokowanej pamięci. Funkcja zwraca informację o miejscu w pamięci gdzie został wpisany obraz. Informacja ta jest wpisywana za pomocą znaku przypisania „=" do wskaźnika *img*.

Funkcja *cvLoadImage* przyjmuje dwa argumenty. Pierwszym jest nazwa pliku z obrazkiem (plik ten musi być zapisany na dysku komputera) oraz parametr sterujący kolorem ładowanego obrazka. Należy zwrócić uwagę na podwojony ukośnik w nazwie pliku. Pojedynczy ukośnik jest w językach C i C++ traktowany jako znak sterujący. Aby kompilator wiedział, że chodzi o ukośnik a nie znak sterujący, ukośnik należy zdublować. Drugi argument funkcji powoduje załadowanie obrazu w odcieniach szarości (gdy jest równy 0) lub w kolorach (gdy jest równy 1).

Kolejne funkcje programu tworzą okienko o nazwie „Obrazek",

```
cvNamedWindow("Obrazek", 1);
```

Wyświetlają obrazek wskazywany przez wskaźnik *img* w okienku „Obrazek",

```
cvShowImage("Obrazek", img);
```

Oczekują na wciśnięcie klawisza,

```
cvWaitKey(0);
```

Zwalniają pamięć zaalokowaną dla obrazka. Od tego momentu wskaźnik img nie powinien być wykorzystywany.

```
cvReleaseImage(&img);
```

Kończą działanie programu.

```
return 0;
```

Zadania A

1. Sprawdź jak zależy działanie programu od drugiego parametru funkcji *cvLoadImage*.

2. Załaduj inny obrazek zmieniając pierwszy argument tej funkcji.

3. Zmień nazwę okienka modyfikując pierwsze argumenty funkcji *cvNamedWindow* i *cvShowImage*.

Dokumentacja funkcji OpenCV

Aby sprawnie wykorzystywać funkcje biblioteczne należy zapoznać się z poszczególnymi funkcjami, zrozumieć ich sposób działania i cel wykonywane przez nie operacji.

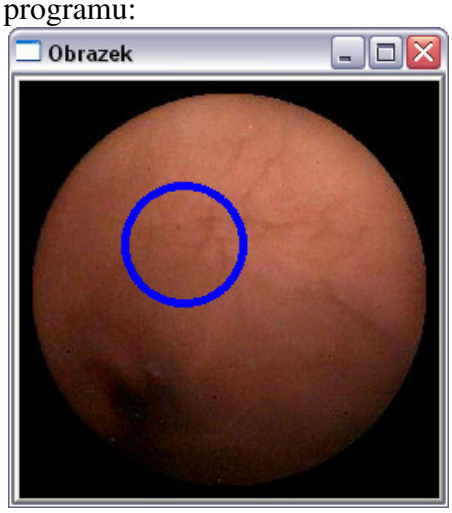
Dokumentacja biblioteki OpenCV w języku angielskim dostępna jest na stronie:
http://opencv.willowgarage.com/wiki/FullOpenCVWiki

Opisy poszczególnych funkcji, pogrupowanych pod kątem ich funkcjonalności i przeznaczenia znajdują się na
http://opencv.willowgarage.com/documentation/index.html

Zadania B

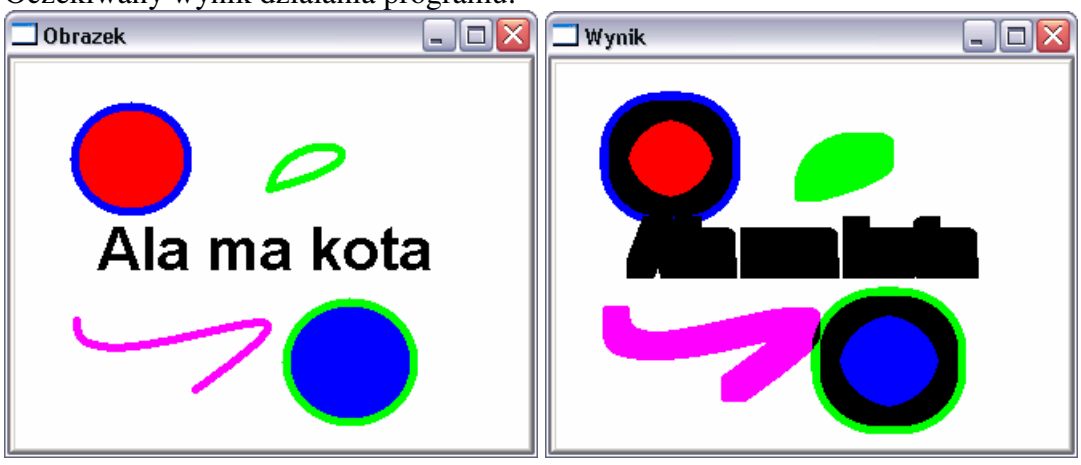
1. W dokumentacji odszukaj informacje o funkcji do rysowania okręgu (cvCircle). Funkcja ta należy do grupy podstawowych (cxcore) służących do rysowania. Dodaj tę funkcję w odpowiednim miejscu programu tak, aby wynikiem działania programu było wyświetlenie obrazka z dorysowanym niebieskim okręgiem. Aby skorzystać z funkcji konieczne jest również wykorzystanie funkcji *cvScalar* i *cvPoint*.

Oczekiwany wynik działania programu:



2. Odszukaj informacje dotyczące funkcji filtrujących. Znajdź opis funkcji erozji *cvErode*. Zmodyfikuj program tak, aby ładował obraz oryginalny, wyświetlał go, tworzył obraz będący wynikiem erozji obrazu oryginalnego i wyświetlał go w osobnym okienku. Do stworzenia drugiego obrazka można wykorzystać funkcję *cvCreateImage*. Pierwszy argument tej funkcji określa rozmiary nowego obrazka. Rozmiary te powinny być identyczne jak rozmiary obrazka oryginalnego. Można je uzyskać za pomocą funkcji *cvSize(img->width, img->height)*, gdzie *img* jest wskaźnikiem do obrazka oryginalnego.

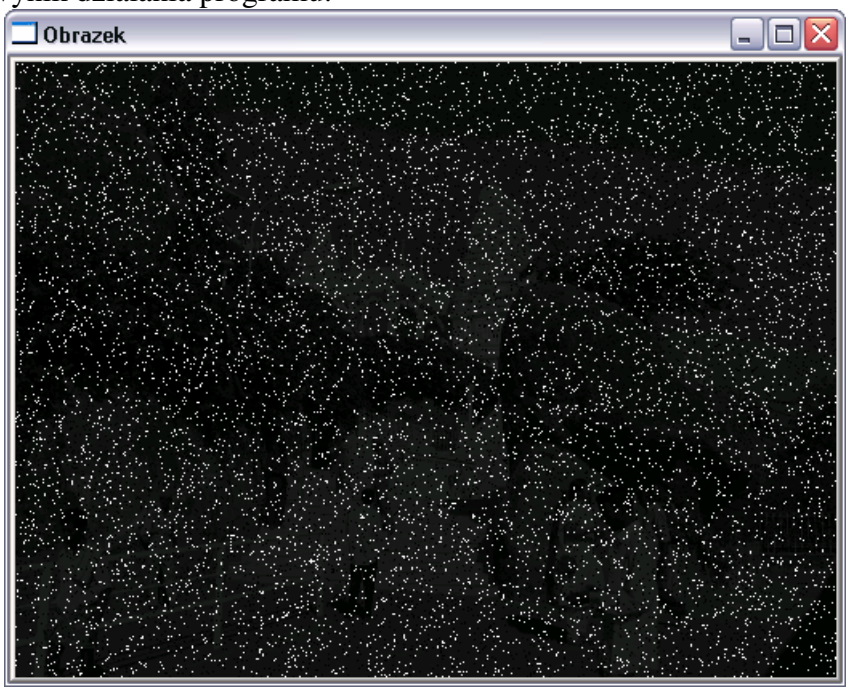
Oczekiwany wynik działania programu:



3. Pobierz plik *Exc2.bmp* ze strony http://www.eletel.p.lodz.pl/pms/dyda_en.html. Napisz program, który usunie z obrazka zakłócenia typu sól i pieprz oraz skoryguje jego histogram.

Do usuwania zakłóceń można wykorzystać filtr medianowy wywoływany funkcją *cvSmooth*. Do korekcji jasności i kontrastu obrazu można wykorzystać funkcję *cvConvertScale*.

Oczekiwany wynik działania programu:



Rozwiązania B

1.

```
int _tmain(int argc, _TCHAR* argv[])
{
	IplImage *img = cvLoadImage("D:\\obrazek.bmp", 1);
	cvCircle(img, cvPoint(100, 100), 36, cvScalar(255, 0, 0), 4);
	cvNamedWindow("Obrazek", 1);
	cvShowImage("Obrazek", img);
	cvWaitKey(0);
	cvReleaseImage(&img);
	return 0;
}
```

2.

```
int _tmain(int argc, _TCHAR* argv[])
{
	IplImage *img = cvLoadImage("D:\\ala.bmp", 1);
	IplImage* wynik = cvCreateImage(cvSize(img->width, img->height),
					IPL_DEPTH_8U, 3);
	cvErode(img, wynik, NULL, 5);
	cvNamedWindow("Obrazek", 1);
	cvNamedWindow("Wynik", 1);
	cvShowImage("Obrazek", img);
	cvShowImage("Wynik", wynik);
	cvWaitKey(0);
	cvReleaseImage(&img);
	cvReleaseImage(&wynik);
	return 0;
}
```

3.

```
int _tmain(int argc, _TCHAR* argv[])
{
	IplImage* orygn = cvLoadImage("D:\\Exc2.bmp", 1);
	IplImage* posrd = cvCreateImage(cvSize(orygn->width, orygn->height),
					IPL_DEPTH_8U, 3);
	IplImage* wynik = cvCreateImage(cvSize(orygn->width, orygn->height),
					IPL_DEPTH_8U, 3);
	cvSmooth(orygn, posrd, CV_MEDIAN);
	cvConvertScale(posrd, wynik, 8, 0);
	cvNamedWindow("Obrazek", 1);
	cvNamedWindow("Wynik", 1);
	cvShowImage("Obrazek", orygn);
	cvShowImage("Wynik", wynik);
	cvWaitKey(0);
	cvReleaseImage(&orygn);
	cvReleaseImage(&posrd);
	cvReleaseImage(&wynik);
	return 0;
}
```