POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Steganografia obrazowa

1. Cel projektu

Celem projektu jest wykazanie, że student nie tylko opanował podstawowe elementy języka C++, ale również ogólnie pojęte umiejętności tworzenia mniejszych projektów programistycznych, na co składają się między innymi umiejętności odpowiedniego doboru narzędzi oraz sporządzanie dokumentacji stworzonych modułów pomocniczych.

Celem projektu nie jest znalezienie i zainstalowanie bibliotek programistycznych.

2. Opis projektu

Projekt polega na stworzeniu aplikacji konsolowej służącej do zapisywania i odczytywania tajnych wiadomości z obrazów komputerowych. Narzędzie to powinno być wygodnym w użyciu modyfikatorem standardowych plików graficznych (takich jak `.png` czy `.bpm`), które jednocześnie ma służyć jako medium odczytywania takich wiadomości.

Wspierane przez nią operacje to: zapisywanie tajnej wiadomości w obrazie komputerowym, odczytywanie tajnej wiadomości z obrazu komputerowego i zwracanie informacji, czy dana wiadomość może zostać zapisana / odczytana z danego pliku.

Aplikacja powinna być uruchamiana z poziomu linii poleceń (ang. *command line*) i, bazując na przekazanych *flagach*, modyfikować, odczytywać lub podawać informacje na temat plików obrazowych.

Dokładna specyfikacja programu znajduje się w dalszych częściach dokumentu.

3. Opis sposobu modyfikacji obrazów

Sposób implementacji steganografii wymagany w tym projekcie opiera się na modyfikacji poszczególnych komponentów obrazów. Student może wybrać dowolne z popularnych formatów obrazu, na przykład `.ppm`, `.jpg` czy `.bmp`, przy czym musi wybrać co najmniej 2. Wspieranie mniejszej liczby rozszerzeń będzie jednoznaczne z niezaliczeniem projektu. Formaty dodatkowo muszą być wspierane przez wiodące systemy operacyjne bez instalowania

dodatkowych narzędzi. Oznacza to, że autorskie lub niewspierane formaty plików będą dyskwalifikowane (chyba że będą one wspierane dodatkowo – poza wymaganymi).

Modyfikacja może (ale nie musi) polegać na zmianie ostatniego bitu każdego piksela mapy binarnej (w przypadku plików, które reprezentują obraz jako ww. mapa) – dzięki temu obraz zostanie nieznacznie zmieniony (najlepiej nierozróżnialnie dla ludzkiego oka), a zapisywana wiadomość może być długa (zależnie od rozmiaru obrazu – obrazy w FullHD mają bardzo dużo takich bitów).

W przypadku bardziej wyrafinowanych formatów (np. polegających na ramkach o zmiennej długości), podejście implementacyjne jest dowolne. Należy jednak optymalizować je pod kątem jak najmniejszych, widocznych zmian w obrazie.

Zabronione są nazbyt trywialne rozwiązania, czyli zapisywanie całej wiadomości w sekcji komentarza całego pliku. Wiadomość ma być trudna do znalezienia dla kogoś, kto nie wie, jak jej ma szukać. W przypadku niejasności, co jest przez to rozumiane, należy skontaktować się z prowadzącym ćwiczenia.

Należy zwrócić uwagę, że niektóre formy zapisu obrazów są bardzo skomplikowane (np. rozszerzenie `.jpg` czy `.gif`) i nie ma prostego sposobu na modyfikowanie ich bez zewnętrznych bibliotek. W związku z tym zaleca się zidentyfikowanie prostszych formatów (np. `.png`, `.ppm` czy `.bmp`). W razie wątpliwości zaleca się skonsultowanie wyborów z prowadzącym ćwiczenia.

4. Wymagania niefunkcjonalne

Projekt powinien być zaimplementowany zgodnie z zasadami produkcji czystego kodu wysokiej jakości oraz zgodny z dobrymi praktykami programistycznymi (zarówno uniwersalnymi jak i stricte dotyczącymi języka C++). Mając na uwadze fakt, że część z tych kryteriów może być kwestią dyskusyjną, student winien upewnić się, jego zrozumienie danych pojęć pokrywa się z elementami przedstawionymi na ćwiczeniach, na przykład poprzez konsultacje z prowadzącym.

Implementacja winna być przemyślana. Oznacza to, że należy dobrać możliwie jak najlepsze narzędzia programistyczne i zastosować je wraz z dobrymi praktykami ich używania. Wynika z tego fakt, że rozwiązanie, które "*po prostu działa*", może nie zdobyć wymaganej do zaliczenia liczby punktów.

Niekompilujące się programy będą automatycznie otrzymywały zero punktów. Należy odpowiednio skorzystać z podzielenia projektu na wiele plików.

Projekt winien być dokumentowany. Wymaganym jest, aby każda nietrywialna funkcja, metoda oraz klasa (ale już nie atrybuty klas) miały dotyczący ich komentarz zgodny ze standardem Doxygen. Głównymi aspektami, na których należy się skupić, są parametry (`@param`), wartości zwracane (`@return`) oraz krótki opis działania (przeznaczenia) danej metody / klasy / funkcji.

W tym opisie nie należy zagłębiać się w szczegóły jak dana idea jest implementowana. Proszę się skoncentrować na opisywaniu tego **co** jest robione, a nie **jak** coś jest robione.

Dobrym przykładem jest chociażby dokumentacja klasy String z Javy, której opis metod można znaleźć tutaj.

W narzędziach typu CLion czy Visual Studio, po zaimplementowaniu funkcji, klasy czy metody, jesteśmy w stanie wygenerować szablon takiej dokumentacji za pomocą wprowadzania następującej sekwencji znaków tuż nad elementem, który chcemy udokumentować: `/**`, a następnie klikając przycisk `Enter`.

5. Wymagania funkcjonalne

Aplikacja powinna obsługiwać przekazane jej argumenty wywołania składające się z flag oraz z danych powiązanych z tymi flagami. Flaga to tekst rozpoczynający się od myślnika i skrótu operacji lub dwóch myślników i pełnej nazwy tej operacji. Wymaganym jest, aby program wspierał poniższy zbiór operacji. Uruchomienie programu z flagą:

- `-i` zakłada, że później zostanie określona ścieżka do pliku. Program winien zbadać, czy ścieżka prowadzi do pliku o obsługiwanym formacie. Na przykład, jeżeli aplikacja wspiera modyfikowanie plików `.png` i `.jpg`, a podana zostanie ścieżka do pliku z rozszerzeniem `.gif` czy `.txt`, program powinien wyświetlić stosowny komunikat o błędzie. W przypadku podania ścieżki do pliku o obsługiwanym formacie, aplikacja winna wyświetlić informacje o tym formacie oraz o samym pliku, takie jak wielkość obrazu, jego zajmowane miejsce w pamięci i timestamp jego ostatniej modyfikacji.

- `-e` zakłada, że później zostanie określona ścieżka do pliku oraz wiadomość. Wiadomość winna być podana w cudzysłowie (aby była traktowana jako pojedynczy argument wywołania). Program ma otworzyć obraz i zapisać w nim sprecyzowaną wiadomość. Podobnie jak w przypadku `-i`, należy zadbać o odpowiednią obsługę błędu, jeżeli ścieżka prowadzi do nieobsługiwanego rodzaju pliku.

- `-d` zakłada, że później zostanie określona ścieżka do pliku, z którego chcemy odczytać wiadomość. Podobnie jak w przypadkach innych flag, należy zadbać o odpowiednią obsługę błędu.

- `-c` zakłada, że później zostanie określona ścieżka do pliku oraz wiadomość. Flaga winna sprawdzić, czy w danym pliku mogłaby zostać zapisana / może istnieć ukryta wiadomość. Potencjalnym problemem może być zbyt mała wielkość pliku (obraz zbyt mały, aby zapisać w nim długą wiadomość).

- `-h`, po której nie oczekujemy żadnych argumentów. Flaga ta winna wydrukować na standardowym wyjściu (najczęściej konsola) informacje o programie, o rozszerzeniach plików obrazowych, które program wspiera oraz sposób użycia i specyfikację reszty flag.

Uruchomienie programu bez sprecyzowania żadnej flagi jest jednoznaczne z uruchomieniem go z flagą `-h`. Uruchomienie programu z nieobsługiwaną (nieznaną) flagą winno skutkować wyświetleniem stosownego błędu i sugestią, aby skorzystać z flagi `-h`. To samo tyczy się użycia flagi z niepoprawnymi argumentami. Zbędne argumenty (np. podanie 5 argumentów po `-h` czy po `-i`) winno być traktowane jako błąd.

Przy każdej operacji polegającej na interakcji z plikami (odczyt / zapis) należy sprawdzić, czy aplikacja ma odpowiednie uprawnienia do takiego polecenia. W przypadku gdy okaże się, że operacja nie może zakończyć się sukcesem, należy wystosować odpowiedni komunikat o błędzie.

Dodatkowo należy obsłużyć aliasy do flag określone przez Tabela 1.

Nazwa flagi	Alias flagi
`-i`	`--info`
`-e`	`--encrypt`
`-d`	`--decrypt`
`-c`	`--check`
`-h`	`--help`

Tabela 1. Aliasy dla flag

Nie przewiduje się obsługi wywołania programu łączącego flagi, tj. nie traktujemy wywołania `xyz.exe -ce [reszta argumentów]` jako `xyz.exe -c [argumenty] -e [argumenty]`.

6. Kryteria oceniania

Tabela 2 określa liczbę punktów do zdobycia za każde z wymienionych wymagań.

Wymaganie	Liczba punktów	Dodatkowy komentarz
Odpowiednie dobieranie i wykorzystywanie narzędzi programistycznych. Poprawne rozdzielenie kodu aplikacji na wiele niezależnych od siebie modułów.	5	Przykładowo używanie standardowych algorytmów zgodnie z ich przeznaczeniem, używanie szablonów zamiast makr, odpowiednie dobieranie kontenerów zgodnie z ich przeznaczeniem. Podział implementacji na wiele plików, stworzenie odpowiednich przestrzeni nazw czy klas. Oczywiście w ramach rozsądku – nic na siłę.

Stosowanie się do *const-correctness*.	1	Oznacza to stosowanie `const` nie tylko wszędzie tam, gdzie można, ale też wszędzie tam, gdzie koncepcyjnie dany element nie powinien być zmieniany.
Unikanie niepotrzebnych kopii danych w programie.	1	Zgodnie z wiedzą przedstawioną na ćwiczeniach. Tyczy się to typów większych niż prymitywy – wskazanym jest kopiowanie zmiennych typu `int` czy `double` zamiast przekazywać je przez `const&` (oczywiście w przypadku niemodyfikowanych kopii oznaczenie ich przez `const` jest wciąż wymagane).
Sporządzenie dokumentacji zgodnej z uproszczonym standardem Doxygen.	4	
Klarowność komunikatów błędów przy niepoprawnym użytkowaniu aplikacji.	5	
Poprawne zaimplementowanie aliasów flag.	1	
Poprawna implementacja flagi `-i`.	2	
Poprawna implementacja flagi `-e`.	10	
Poprawna implementacja flagi `-d`.	10	
Poprawna implementacja flagi `-c`.	3	
Poprawna implementacja flagi `-h`.	2	
Trudność wspierania wybranych rodzajów plików obrazowych (wraz z ich liczbą) .	6	

Tabela 2. Kryteria oceniania

7. Obrona

Student będzie przepytany ze szczegółów implementacyjnych swojego rozwiązania. Brak zrozumienia i identyfikacji dowolnego fragmentu kodu może być podstawą do **niezaliczenia całego projektu**. W związku z tym sugeruje się, aby studenci, którzy ukończyli projekt semestralny długo przed terminem obrony, zadbali o to, aby wszystkie szczegóły projektu zostały przypomniane.

8. Dodatkowe uwagi

Zabrania się stosowania rozwiązań, których student nie rozumie w dobrym stopniu. Każde wykorzystane narzędzie musi być opanowane przez studenta. Zezwala się na używanie elementów, które nie były przedstawione na ćwiczeniach czy wykładzie, lecz należy równocześnie pamiętać, że odpytywanie podczas obrony może się skoncentrować również i na tych elementach.

W przypadku dowolnych niejasności należy skonsultować treść i wymagania projektu z prowadzącym ćwiczenia.