
POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

# PJAText2

## 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 programistycznych oraz sporządzanie dokumentacji zbudowanych modułów pomocniczych.

## Opis projektu

Projekt polega na stworzeniu aplikacji o nazwie PJAText2. Narzędzie to winno być wygodnym w użyciu programem, który służyć będzie do przeróżnych operacji na zawartościach plików tekstowych zapisanych w formacie ANSI (tylko znaki ASCII).

Wspierane przez nią operacje to, między innymi: wyświetlanie liczby słów w pliku, wyświetlanie liczby liczb w pliku, wyświetlanie liczby linii w pliku, porządkowanie elementów z tego pliku czy przeszukiwanie zawartości pliku w celu znalezienia elementów spełniających dane kryteria.

Aplikacja winna być uruchamiana z poziomu linii poleceń (ang. *command line*) i, bazując na przekazanych flagach, produkować odpowiedni tekst zawierający zgodne z wymaganiami informacje.

Dokładna specyfikacja programu znajduje się w dalszych częściach dokumentu.

Filip Kwiatkowski, fk@pjwstk.edu.pl

# Wymagania niefunkcjonalne

Projekt winien być zaimplementowany zgodnie z zasadami produkcji czystego kodu oraz zgodnie 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, że jego lub jej 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, iż rozwiązanie, które "po prostu działa" może nie zdobyć wymaganej do zaliczenia liczby punktów.

# 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:

1. `-f` winno oczekiwać, że następnym argumentem będzie ścieżka do **pliku źródłowego**. W przypadku niepoprawnej ścieżki lub jej braku program powinien wyświetlić odpowiedni, przyjazny dla użytkownika komunikat o błędzie.
2. `-n` winno wyświetlić liczbę linii z pliku źródłowego. Naturalnie oznacza to, że flaga ta winna być precyzowana tylko jeżeli obecna jest flaga `-f`. W przypadku braku takowej użytkownik oczekuje stosownego i zrozumiałego komunikatu z błędem.
3. `-d` winno wyświetlić liczbę cyfr z pliku źródłowego. Cyfra nie musi być osobnym słowem (czyli oddzielona białymi znakami lub początkiem / końcem pliku).
4. `-dd` winno wyświetlić liczbę liczb z pliku źródłowego. Liczby muszą być oddzielone białymi znakami lub początkiem / końcem pliku. Mówiąc krótko, liczba to słowo, które składa się ze znaków, które możemy całościowo zinterpretować jako **liczbę rzeczywistą**.
5. `-c` winno wyświetlić liczbę znaków z pliku źródłowego.
6. `-w` winno wyświetlić liczbę słów z pliku źródłowego. Słowo to dowolny ciąg niebiałych znaków. Liczby również traktujemy jako słowo.

7. `-s` winno wyświetlić wszystkie słowa z pliku wejściowego w kolejności alfabetycznej.
8. `-rs` winno wyświetlić wszystkie słowa z pliku wejściowego w kolejności odwrotnej od alfabetycznej.
9. `-l` winno modyfikować działanie następującej po niej fladze `-s` lub `-rs` tak, aby zamiast brać pod uwagę kolejność alfabetyczną, flagi te brały pod uwagę kolejność precyzowaną przez **długość** poszczególnych słów.
10. `-a` winno być **ostatnią** sprecyzowaną flagą, po której nastąpi nieokreślona liczba słów (może być 0). Program z tak podaną flagą winien wyświetlać wszystkie słowa z pliku źródłowego, które są anagramami słów podanych tuż po tej fladze. Tak wyświetlane słowa nie powinny się powtarzać (choć słowa podane po fladze `-a` mogą się powtarzać).
11. `-p` winno być **ostatnią** sprecyzowaną flagą, po której nastąpi nieokreślona liczba słów (może być 0). Program z tak podaną flagą winien wyświetlać wszystkie podane po fladze `-p` słowa, które są palindromami **oraz występują w pliku źródłowym**. Tak wyświetlane słowa nie powinny się powtarzać (choć słowa podane po fladze `-p` mogą się powtarzać).
12. `-o`, po której oczekuje się ścieżki do pliku, winno modyfikować działanie programu w taki sposób, że zamiast wyświetlać informacje na konsoli, w której uruchamiana jest aplikacja, efekt działania programu w postaci tekstu został **zapisany do pliku** o sprecyzowanej (wcześniej wspomnianej) ścieżce. Jeżeli taki plik nie istnieje, to **powinien zostać stworzony**.
13. `-i`, po której oczekuje się ścieżki do **pliku wejściowego**, winno modyfikować działanie programu w taki sposób, że zamiast precyzować resztę flag jako argumenty wywołania, aplikacja winna pobrać flagi i dane ich dotyczące z wymienionego wcześniej pliku wejściowego. Flaga ta winna być jedyną podaną podczas uruchamiania aplikacji (reszta flag znajduje się w pliku wejściowym). W przypadku, kiedy flaga `-i` jest obecna, ale nie jest jedyna, program powinien wystosować odpowiedni komunikat o błędzie.

Dodatkowo, uruchomienie programu bez żadnej flagi winno skutkować wyświetleniem informacji o tym, co to jest za program i jakie opcje (wraz z jakim użyciem) są wspierane.

Aplikacja powinna wpierw weryfikować, czy podane flagi wykonania oraz ich dane są poprawne. Oczekuje się, że dowolny błąd będzie skutkować odpowiednim komunikatem, a nie wykonaniem pracy związanej z części flag, które są sprecyzowane przed błędną flagą / danymi.

Implikuje to, że pojedyncze uruchomienie programu winno wspierać dowolną **permutację** flag, chyba że specyfikacja którejś z nich tego zabrania. Przykładowo, aplikacja powinna poprawnie obsłużyć sprecyzowane flagi `-f` ze ścieżką, trzykrotne sprecyzowanie flagi `-s`, po której nastąpią flagi `-l`, `-rs` i `-s`, a na koniec flaga `-c`. Takie wywołanie powinno skutkować trzykrotnym wyświetleniem się wszystkich, uporządkowanych alfabetycznie słów w pliku wejściowego, następnie wyświetlać wszystkie słowa od najdłuższego do najkrótszego (`-l`, czyli po długości i `-rs`, czyli odwrotnie od domyślnej (rosnącej)), po czym znowu wyświetlać wszystkie słowa alfabetycznie (poprzednie `-l` zostało "pochłonięte" przez wykonane `-rs`) i na koniec oczekujemy liczby reprezentującej liczbę znaków z pliku. Jak widać, kolejność precyzowania flag ma znaczenie jeżeli chodzi o output programu.

Uruchomienie programu z flagą, której na powyższej liście nie ma, winno skutkować wyłącznie odpowiednim komunikatem o błędzie z tego wynikającym. Jeżeli nie zostało to sprecyzowane przy konkretnym punkcie, to w przypadku niepoprawnego użycia flagi (na przykład flaga `-a` niebędąca ostatnią flagą), należy obsłużyć błąd w stosowny, według studenta, sposób. Interpretacja tej informacji jest dowolna, lecz zakładamy, że jakaś obsługa błędu i **stosowny komunikat** muszą mieć miejsce.

Dodatkowo należy obsłużyć aliasy do flag zaprezentowane przez poniższą tabelę (Tabela 1).

| Nazwa flagi | Alias flagi |
|---|---|
| `Brak flagi` | `--help` |
| `-f` | `--file` |
| `-n` | `--newlines` |
| `-d` | `--digits` |
| `-dd` | `--numbers` |
| `-c` | `--chars` |
| `-w` | `--words` |
| `-s` | `--sorted` |
| `-rs` | `--reverse-sorted` |

| | |
|---|---|
| `-l` | `--by-length` |
| `-a` | `--anagrams` |
| `-o` | `--output` |
| `-p` | `--palindromes` |
| `-i` | `--input` |

*Tabela 1*

Dodatkowo, każdy ze studentów winien wprowadzić własną flagę (oraz jej alias). Jej działanie może w jakiś sposób modyfikować działanie innych flag lub wprowadzać zupełnie nową funkcjonalność.

Projekt winien być również udokumentowany. Wymaganym jest, aby każda 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. Dobrym przykładem jest chociażby dokumentacja klasy String z Javy, której opis metod można znaleźć tutaj.

## Kryteria oceniania

Poniższa tabela (Tabela 2) określa liczbę punktów do zdobycia za każde z wymienionych wyżej wymagań.

| Wymaganie | Liczba punktów do zdobycia | Dodatkowy komentarz |
|---|---|---|
| Wyświetlenie odpowiedniego komunikatu w przypadku braku jakiejkolwiek flagi. | 4 | Tyczy się to również wywołania programu z flagą -i z poprawną ścieżką, ale z pustym plikiem wejściowym. |
| Traktowanie pliku wskazanego przez ścieżkę sprecyzowaną po fladze -f jako pliku źródłowego. | 4 | |

| | | |
|---|---|---|
| Wyświetlenie liczby linii w pliku w przypadku sprecyzowania flagi `-n`. | 4 | |
| Wyświetlenie liczby cyfr w pliku w przypadku sprecyzowania flagi `-d`. | 4 | |
| Wyświetlenie liczby liczb w pliku w przypadku sprecyzowania flagi `-dd`. | 4 | Tyczy się to również liczb zmiennoprzecinkowych. |
| Wyświetlenie liczby słów w pliku w przypadku sprecyzowania flagi `-w`. | 4 | |
| Wyświetlenie wszystkich słów z pliku w kolejności alfabetycznej w przypadku sprecyzowania flagi `-s`. | 4 | |
| Wyświetlenie wszystkich słów z pliku w kolejności odwrotnej od alfabetycznej w przypadku sprecyzowania flagi `-rs`. | 4 | |
| Modyfikacja działania następnej flagi `-s` lub `-rs` w taki sposób, aby kolejność słów była determinowana nie przez porządek alfabetyczny, a przez ich długość w przypadku sprecyzowania flagi `-l`. | 4 | Flaga `-l` winna mieć wpływ tylko na **następną** flagę `-s` lub `-rs`. Kilka podanych flag `-l` obok siebie należy traktować jako jedną, pojedynczą flagę. |
| W przypadku podania flagi `-a`, weryfikacja, czy flaga ta jest podana jako ostatnia, poprawne sczytanie wszystkich słów po niej następujących i wyświetlenie (bez powtórzeń) wszystkich anagramów tych słów z pliku. | 4 | Zakładamy, że w pliku nie ma słów wyglądających jak flagi – jeżeli słowo podane po `-a` wygląda jak flaga, powinno być traktowane jako flaga i aplikacja winna zakomunikować odpowiedni błąd. |

| | | |
|---|---|---|
| Przekierowanie outputu do sprecyzowanego pliku o ścieżce sprecyzowanej tuż po fladze w przypadku flagi `-o`. | 4 | |
| W przypadku podania flagi `-p`, weryfikacja, czy flaga ta jest podana jako ostatnia, poprawne sczytanie wszystkich słów po niej następujących i wyświetlenie (bez powtórzeń) tych podanych słów, które są palindromami i występują w pliku. | 4 | Zakładamy, że w pliku nie ma słów wyglądających jak flagi – jeżeli słowo podane po `-p` wygląda jak flaga, powinno być traktowane jako flaga i aplikacja winna zakomunikować odpowiedni błąd. |
| Modyfikacja działania programu w taki sposób, że reszta flag winna być pobrana z pliku sprecyzowanego przez ścieżkę podaną tuż po fladze `-i`. | 4 | |
| Poprawna obsługa własnoręcznie zaprojektowanej flagi o wybranej nazwie. | 4 | Opis działania flagi winien być umieszczony w kodzie źródłowym jako komentarz. |
| Poprawna obsługa aliasów do flag. | 4 | Dotyczy to również własnoręcznie zaprojektowanej flagi przez studenta. |
| Odpowiednie dobieranie i wykorzystywanie narzędzi programistycznych. | 5 | Przykładowo używanie standardowych algorytmów zgodnie z ich przeznaczeniem, raczej używanie szablonów niż makr, korzystanie z `std::function` zamiast ze wskaźników na funkcje, odpowiednie dobieranie kontenerów zgodnie z ich przeznaczeniem.<br><br>Dobieranie i stosowanie narzędzi tak, aby jasno wskazywały intencje programistyczne – zawiera się |

| | | |
|---|---|---|
| | | w tym również czysty kod, a zatem i wybieranie deskryptywnych, dobrych nazw zmiennych. |
| Stosowanie się do *const-correctness*. | 5 | 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. | 5 | Zgodnie z wiedzą przedstawioną na ćwiczeniach, tyczy się to typów większych niż prymitywy – wskazanym jest kopiowanie zmiennych typu int czy double zamiast przekazywania ich przez `const&` (oczywiście w przypadku niemodyfikowanych kopii oznaczenie ich przez `const` jest wciąż wymagane). |
| Sporządzenie dokumentacji zgodnej z uproszczonym standardem Doxygen. | 15 | |

*Tabela 2*

Warto również dodać, że kolejność flag (o ile nie jest to sprecyzowane inaczej) nie ma znaczenia. Na przykład w przypadku sprecyzowania kilku flag, wśród których **gdzieś w środku** będzie sprecyzowana flaga `-o`, **całość** outputu winna być przekierowana do pliku wyjściowego.

Przykładowym odstępstwem od takiej dowolnej kolejności podawania flag jest flaga `-l`, która aplikuje się **tylko do następnej** flagi będącej flagą `-s` lub `-rs`.

## 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.

## 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 przepytywanie może się skoncentrować również i na tych elementach.