SPRAWOZDANIE

PIOTR ROGOWSKI 192258

# Cel projektu

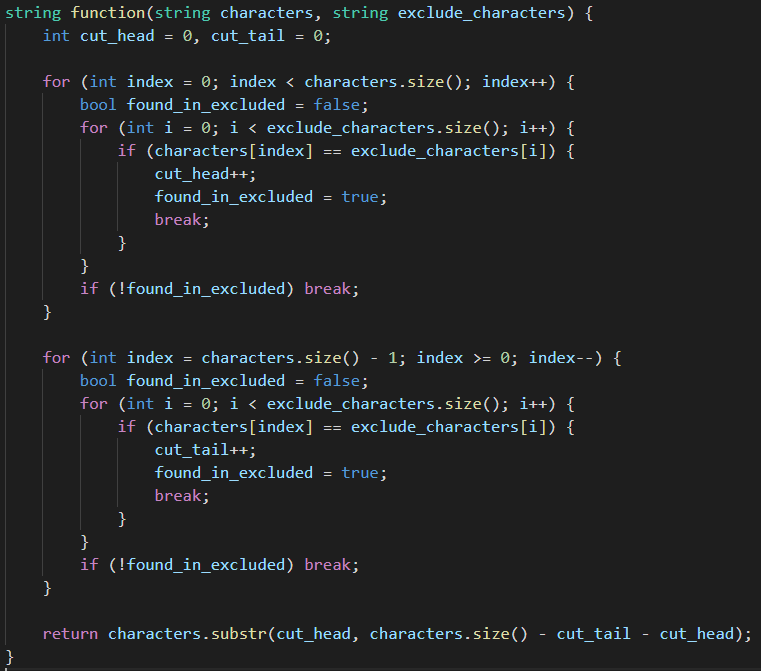
Celem projektu jest zaimplementowanie dowolnej funkcji zawartej w bibliotece standardowej języka Python. Funkcja, którą wybrałem to *str.strip()*. Jako argument przyjmuje jedną, opcjonalną wartość w formie ciągu znaków, które zostaną usunięte z początku oraz końca wskazanego ciągu znaków. Przykład użycia funkcji: „piotr”.strip(„pro”) = „iot”. W przypadku niepodania argumentu funkcja usuwa białe znaki z początku i końca podanego ciągu znaków.

# Implementacja funkcji

* 1. Język C++

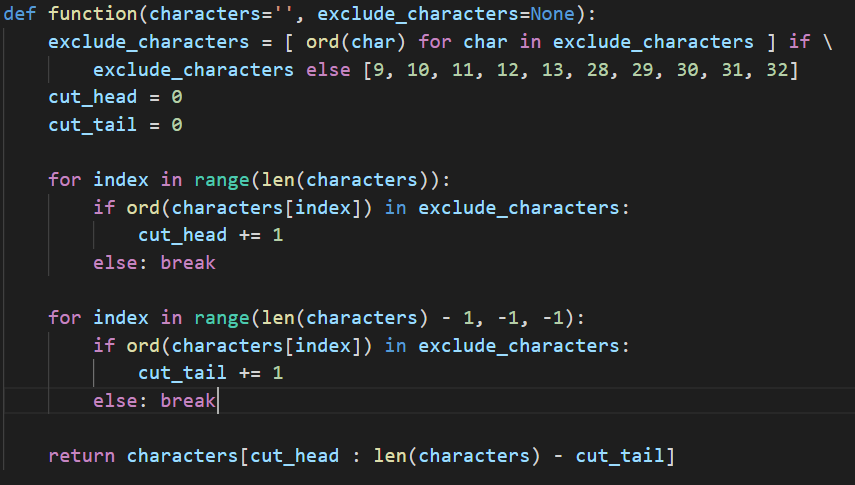


**Rys 1.** Fragment pliku nagłówkowego zawierający deklarację funkcji m.in. wartość opcjonalnego argumentu



**Rys 2.** Kod źródłowy własnej implementacji funkcji w języku C++

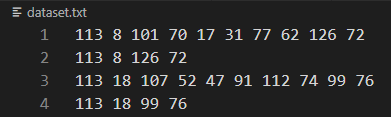
* 1. Język Python



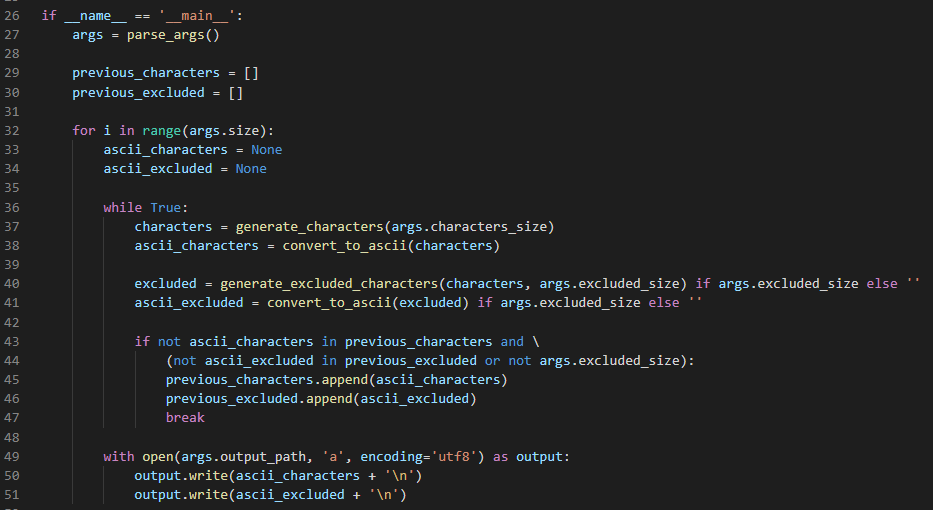
**Rys 3.** Kod źródłowy własnej implementacji funkcji w języku Python

# Wygenerowane dane

Na potrzeby projektu wygenerowano 2000 unikalnych danych. Format zapisu danych to wartości kodów z przedziału [0; 127], który przyjęto ze względu na sposób kodowania znaków przez typ *string* w języku C++. Typ *string* traktuje pojedynczą wartość znaku jako *signed char*, przez co jest on w stanie przyjąć wartości z przedziału [-128, 127]. W instrukcji zawarta jest informacja, aby przyjąć założenie w formie znaków z przedziału [0, 255], co jest równoznaczne typem *unsigned char*. Jednak przy wpisaniu typu *unsigned char* do typu *string* staje on się typem char ze znakiem. Opisane powyżej uproszczenie zostało skonsultowane z prowadzącym oraz zaakceptowane. Połowa wygenerowanych danych zawiera opcjonalny parametr. Pojedynczy generowany przypadek został wygenerowany w dwóch liniach pliku tekstowego. W linii nieparzystej zapisane są kody znaków, dla których wywoływana będzie funkcja. W liniach parzystych zapisane zostały kody znaków podawane jako opcjonalny parametr. Przypadki testowe niezawierające opcjonalnego parametru posiadają linię parzystą pustą. Poniżej przedstawiono fragment pliku z wygenerowanymi danymi.



**Rys 4.** Fragment pliku z danymi



**Rys 5.** Fragment kodu umożliwiający generowanie danych testowych

# Mierzenie czasu

* 1. Funkcje

W implementacji funkcji napisanej w języku Python do pomiaru czasu użyta została funkcja *time.time\_ns()*. Funkcja zwraca czas, który upłynął w nanosekundach. Jako dokładność pomiaru czasu przez funkcję *time.time\_ns()* przyjęto wartość 1 sekundy.

W implementacji funkcji napisanej w języku C++ do pomiaru czasu użyta została funkcja *now()* będąca częścią klasy *steady\_clock* biblioteki *chrono*. Funkcja zwraca czas, który upłynął w dowolnej jednostce czasu dzięki rzutowaniu na jednostkę czasu. Jako dokładność pomiaru czasu przez funkcję również przyjęto wielkości 1 sekundy.

* 1. Sposób

Pomiar pojedynczego wywołania funkcji jest mniejszy niż założony błąd pomiaru, przez co nie jest to wiarygodne. Istnieje zatem potrzeba mierzenia czasu wykonania większej ilości wywołań funkcji. Ten sposób jednak wiąże się z dodatkowym czasem, jaki kompilator oraz interpreter muszą poświęcić na iterację po kolejnych wywołaniach funkcji. Aby policzyć tylko czas czystych obliczeń od czasu obliczeń w pętli potrzeba odjąć czas trwania pustej pętli. Zatem czas mierzony zostanie 3 razy: przed wywołaniem pustej pętli, po wywołaniu pustej pętli oraz po wywołaniu pętli z obliczeniami. Różnice pomiędzy poszczególnymi czasami są kolejno równe czasom trwania pustej pętli oraz pętli z obliczeniami. W celu uzyskania czasu samych obliczeń wystarczy obliczyć różnicę pomiędzy tymi wartościami.

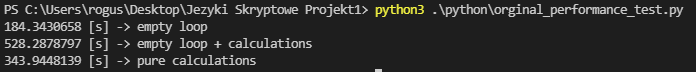
W instrukcji wspomniany został także błąd względny, który ma mieć wartość nieprzekraczającą 1%. Zatem mając do wykonania 3 pomiary czasu (każdy z błędem pomiaru w wysokości 1 sekundy) przeprowadzone eksperymenty powinny trwać minimum 300 sekund, aby błąd względny był mniejszy niż 1%.



**Rys 6.** Fragment kodu w języku C++ przedstawiający sposób pomiaru czasu

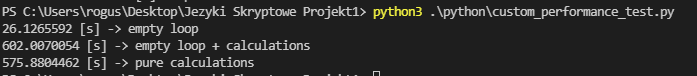
# Test wydajności

Test wydajności funkcji str.strip():



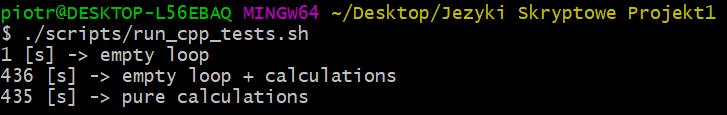
**Rys 7.** Wynik wywołania testu wydajności funkcji str.strip()

Implementacja własnej funkcji w Pythonie:



**Rys 8.** Wynik wywołania testu wydajności autorskiej funkcji w języku Python

Implementacja własnej funkcji w C++:



**Rys 9.** Wynik wywołania testu wydajności autorskiej funkcji w języku C++

Podsumowanie testów wydajności dla wszystkich implementacji:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Implementacja** | **Czas [s]** | **Błąd względny [%]** | **Ilość powtórzeń (n)** | **Średni czas jednego wywołania funkcji [ns]** |
| str.strip() | 344 | 0,872093023 | 1 500 000 | 114,7 |
| Python | 576 | 0,520833333 | 200 000 | 1440 |
| C++ | 435 | 0,689655172 | 300 000 | 725 |

**Tabela 1.** Tabela wyników pomiaru wydajności oryginalnej funkcji z autorskimi w językach Python i C++

Najszybsza w działaniu okazała się funkcja biblioteki standardowej *str.strip()*. Kolejną najszybszą jest implementacja własna funkcji w języku C++. Dwa razy wolniejszą od implementacji w języku C++ okazała się implementacja własna w języku Python.

# Test implementacji

W celu walidacji zwracanych przez wszystkie implementacje wyników zapisywane są one do plików tekstowych w formacie ciągów znaków. Napisany został odpowiedni skrypt w języku Python, który pozwala na wczytanie zawartości plików oraz ich porównanie.



**Rys 10.** Kod w języku Python realizujący funkcjonalność testowania implementacji

# Podsumowanie

Najszybszą implementacją okazała się oryginalna implementacja standardowej biblioteki Pythona. Najprawdopodobniej działanie algorytmu zostało staranie przemyślane tak, aby możliwie zoptymalizować czas jego działania. Natomiast zaprojektowany przeze mnie algorytm odbiega pod kątem złożoności czasowej od oryginalnej implementacji. Różnica czasów wykonania algorytmu w języku C++ oraz Python najprawdopodobniej wynika z charakteru obu języków. Język C++ jest kompilowany do kodu maszynowego bezpośrednio wykonywanego przez procesor natomiast Python jest językiem interpretowanym co może mieć znaczny wpływ na czas wykonania obu programów. Również Python jest językiem dynamicznie typowanym co oznacza że wnioskuje on typy przechowywanych zmiennych w trakcie wykonywania programu, natomiast C++ jest językiem ze statycznym typowaniem zmiennych. Różnica ta również może mieć wpływ na wydajność implementacji w obu językach.