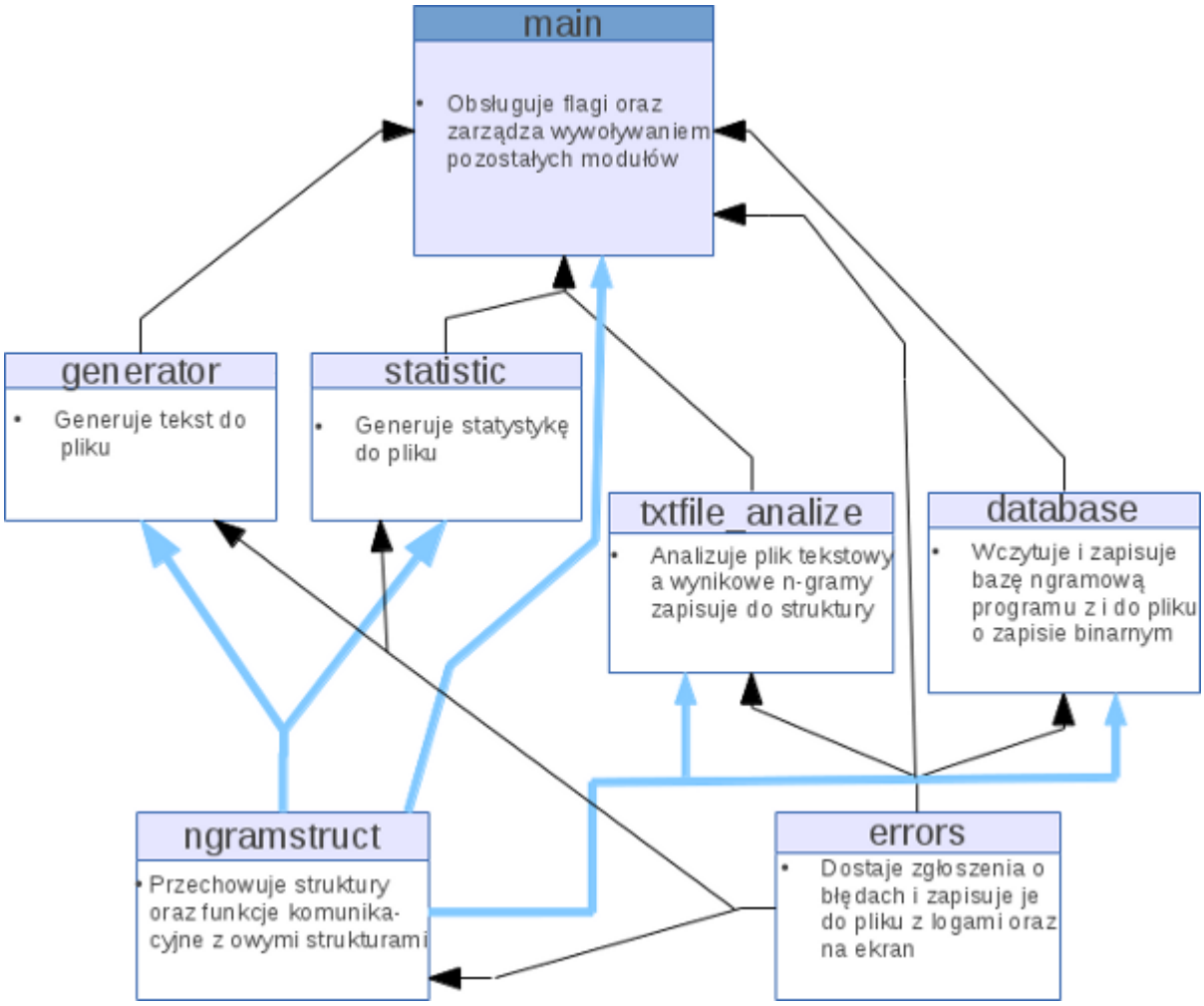
Specyfikacja implementacyjna programu "Wyprodukuj tekst".

Projekt na zajęcia: *"Języki i Metodyka Programowania"*
wydz. Elektryczny Politechniki Warszawskiej, rok akademicki 2014/2015
wykonał: Paweł Drapiewski

1. Podział na moduły



Powyższy diagram przedstawia schematyczną strukturę plików z jakich składa się program "Produkuj Tekst". Pojedynczy prostokąt odzwierciedla moduł który składa się z pliku nagłówkowego oraz pliku z kodem źródłowym.

2. Szczegółowy opis modułów

2.1 main

Głównym zadaniem tego modułu jest rozpoznanie flag, i na podstawie ich interpretacji sterowanie całym programem. To ten moduł będzie miał dostęp do wszystkich pozostałych modułów. Dodatkowo korzystać on będzie z biblioteki zewnętrznej unistd.h.

Ważniejsze funkcje i zmienne:

- **static int check_flags(int argc, char ** argv)** - funkcja która rozpoznaje flagi korzystając z biblioteki unistd
- **enum err_codes { ERR_IN_FLAGS, ERR_IN_INPUT_TXT_FILE, ... }** - typ wyliczeniowy przypisujący kod błędu do miejsca wystąpienia błędu, który później będzie używany do zapisu go w logach, a to wszystko by łatwo móc nimi zarządzać w przyszłości.

- **void main(int argc, char **argv)** - główna funkcja całego programu, to z jej poziomu będzie odbywało się sterowanie

2.2 generator

Jego zadaniem będzie główny założenie programu czyli generacja tekstu wprost do pliku, na podstawie utworzonej wcześniej bazy słów.

Ważniejsze funkcje i zmienne:

- **int generate(ngram * ngramstack, char * start_word, int max_word, char * file_path)** - główna funkcja generująca, gdzie jako **ngramstack** to wskaźnik na bazę danych z ngramami, **start_word** to łańcuch tekstowy od którego rozpoczynamy generacje (gdy pusty to będziemy losować z istniejącego zbioru), **max_word** określa z ilu słów ma się składać wygenerowany tekst, a **file_path** określa ścieżkę zapisu pliku.

Ważniejsze algorytmy:

Algorytm generowania tekstu z zastosowaniem łańcuchów Markova:

1. Gdy nie posiadamy ngramu wzorcowego podanego przez użytkownika to losujemy go z naszej bazy danych
2. Dla danego ngramu wywołujemy funkcję **ngram_find_sufixs**, gdzie jako parametry podajemy nasz sufiks, a w rezultacie otrzymamy macierz dostępnych sufiksów, Macierz przypisujemy macierzy tmp
3. Wylosuj jedno ze słów z macierzy tmp, wylosowane słowo zapisz do pliku, a gdy macierz pusta zakończ działanie
4. zmień aktualnie przetwarzany ngram, usuwając pierwszy wyraz prefiksu oraz dodając na koniec wylosowany sufiks
5. gdy nie osiągnięto wymaganej ilości słów wygenerowanych wróć do kroku pierwszego z nowym ngramem wejściowym

2.3 ngramstruct

Jeden z ważniejszych modułów programu, gdyż opisuje struktury przechowujące dane oraz funkcje dostępowe do tych struktur, oraz funkcję operujące na tych strukturach, które to są niezbędne dla działania pozostałych modułów (np. ngram_search).

Zastosowane struktury:

- struktura pomocnicza, przechowująca słowa wczytane z pojedyńczego pliku wejściowego

```
struct word_collect {
    char ** words;
    char * name_file; //nazwa pliku z ktorego pochodza slowa
    int * num_words; //liczba wczytanych slow
}
```

- struktura przechowująca zbiór wszystkich słów wczytanych do programu, będąca główną strukturą programu

```
typedef struct {
    struct word_collect * one_file;
    int ngram_size; //rozmiar zaalokowany zostal na tyle elementow
    int ngram_elem; //tyle elementow jest aktualnie w tablicy
} ngram;
```

- struktura dla stringów, by w prosty sposób można było je przekazywać pomiędzy funkcjami programu

```
typedef struct str {
    char * field;
    struct str * next;
} string_stock;
```

Ważniejsze funkcje i zmienne:

- **void string_stock_push(string_stock where, char * what)** - odkłada string'a na stos
- **char * string_stock_pop(string_stock from)** - zdejmuje string ze stosu, i "oddaje" go poprzez wartość zwracaną.
- **string_stock * string_stock_init()** - inicjuje stos zanim będzie można coś na niego odłożyć, zwraca wskaźnik na zainicjowany stos
- **static string_stock * string_stock_resize(string_stock from)** - gdy będzie brakować pamięci to zostanie ona powiększona, w razie niepowodzenia zwróci pusty wskaźnik, w innym przypadku będzie to wskaźnik na powiększony stos
- **ngram * ngram_init()** - inicjuje strukturę i zwraca wskaźnik na nią
- **int ngram_add(ngram * this, char ** words_collect, int num_words, char * file_name)** - dodaje do struktury **this** pojedyńczą komórkę z strukturą word_collect, która przechowuje tablicę słów **words_collect**, ilość tych słów **num_words** oraz nazwę pliku z którego pochodzi **file_name**
- **int ngram_free(ngram * this)** - uwalnia pamięć zajmowaną przez daną strukturę
- **char ** ngram_find_sufixs(ngram structure, char ** prefix, int num_prefix)** - przeszukuje całą strukturę ze zbiorem wszystkich słów i na podstawie prefixów odnajduje sufix'y, które składa w tablicę i zwraca na wyjściu. Gdy nie znajdzie żadnego to zwraca NULL. Sam **num_prefix** oznacza po prostu z ilu słów składa się prefix.

2.4 statistic

Moduł ten tworzy statystyki i zapisuje je do danego pliku

Ważniejsze funkcje i zmienne:

- **int statistics(ngram * ngramstack, char * fraza, char * file_path)** - funkcja odpowiedzialna za sterowanie statystykami i zapis ich do pliku.

- **static int teor_prob(ngram * ngram_stack, char * phrase)** - wylicza prawdopodobieństwo teoretyczne, czyli bada każde słowo z phrase pod względem ilości wystąpień i dzieli przez ogólną liczbę wszystkich słów, a jako wynik daje iloczyn tych liczb

- **static int real_prob(ngram * ngram_stack, char * phrase)** - oblicza jakie było rzeczywiste prawdopodobieństwo kolokacji słów zapisanej w phrase w następujący sposób: Wpierw liczy ile razy wystąpiła cała fraza we wszystkich ngramach, a następnie liczy ile można by utworzyć takich fraz na podstawie naszej bazy. Sposób takiego liczenia jest następujący: dla każdego zbioru słów umieszczonego w pojedynczej komórki struktury liczy się liczbę możliwych ngramów według wzoru ilość_słów - (ilość_słów_w_frazie + 1), i takie liczby możliwych ngramów się sumuje dla każdej komórki struktury. Funkcja na koniec zwraca liczbę wystąpień frazy dzieloną przez liczbę możliwych ngramów.

- **int PMI(ngram * ngramstack, char * fraza)** - liczy PMI (Pointwise Mutual Information) czyli współczynik pomiędzy wartością oczekiwaną a uzyskaną, liczona jest następująco ln(**teor_prob** / **real_prob**).

2.5 database

Ten moduł zajmuje się obsługą baz danych, czyli zapisem i odczytem struktur ngram z pliku binarnego.

Ważniejsze funkcje i zmienne:

- **int save_ngram(ngram * ngramstack, char * file_path)** - zapisuje aktualną bazę danych do pliku

- **int read_ngram(ngram * ngramstack, char * file_path)** - odczytuje z pliku bazę danych i dodaje do bazy danych programu.

2.6 textfile_analize

Moduł ten służy do analizy plików tekstowych, które zawierają wzorcowy tekst dla naszego generatora.

Ważniejsze funkcje i zmienne:

- **int analize(ngram * ngramstack, char * file_path)** - główna funkcja tego modułu, analizuje ona plik wejściowy. A jej działanie jest następujące: tworzy ona tablicę w której będzie przechowywała słowa, następnie będzie każde napotkane słowo dodawać do tej tablicy, aż nie napotka końca pliku. I tak przygotowaną tablicę podepnie pod strukturę bazy danych, jako nowe pole.

2.7 errors

Jego zadaniem jest obsługa błędów, czyli wypisanie błędu na ekran oraz do logów i zakończenie aplikacji w przypadku błędu krytycznego.

Ważniejsze funkcje i zmienne:

- **void error(int err_type, int err_code, char err_content[])** - spełnia założenia przedstawione w opisie modułu. A same argumenty mają się następująco: **err_type** określa typ błędu, i dla wartości równej 1 (krytyczna) wyłącza program, **err_code** stanowi cyfrę informujące gdzie nastąpił błąd, kod ten będzie zapisywany w logach by ułatwić administracje tymi danymi, za to **err_content**[] jest tablicą z komunikatem słownym, który dokładnie określa parametry błędu.

3. Testowanie

Testowanie programu będzie w pierwszym etapie polegało na napisaniu dokładnych testów dla każdego z modułu osobno, i dopiero po pozytywnym przejściu tych testów rozpocznie się test całego programu przy pomocy wprowadzania różnych zestawów danych, m.in. generacja na podstawie dzieł literackich. Gdy program będzie już generował poprawne dane, będzie można zająć się naprawą mniej istotnych elementów jak wycieki pamięci. Testy te będą wykonane przy pomocy programu **valgrid**. Dodatkowo do programów testujących można zaliczyć GNU Debugger, który będzie towarzyszył całemu okresowi budowy programu, gdyż jest nieodzownym narzędziem do wyszukiwania błędów.