SPECYFIKACJA IMPLEMENTACYJNA

Paweł Skiba, Jakub Świder

30 marca 2019

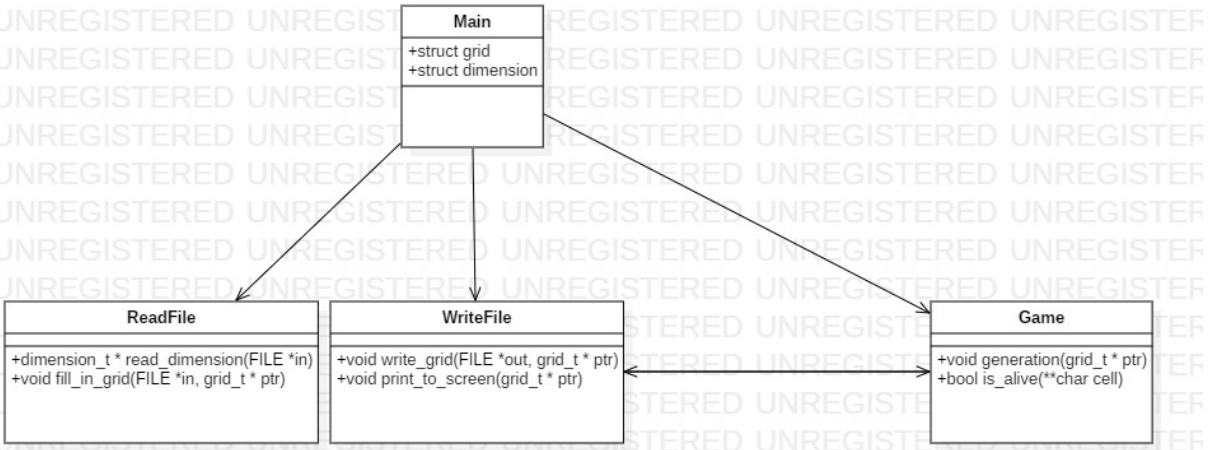
Spis treści

1 Wstęp

Program Game Of Life będzie wywoływany domyślnie z trzema zmiennymi: liczbą generacji do wygnerowania, nazwą pliku z planszą wejściową oraz nazwą pliku do którego będą zapisywane kolejne generacje. Do przekazania argumentów wywołania służyć będą flagi. W ten sposób kolejność ich wypisywania po nazwie programu nie będzie miała znaczenia. Dzięki flagom możliwa również będzie zmiana znaków symbolizujących komórki żywe i martwe. Dostępne będą następujące flagi:

- -n - określa liczbę generacji do wytworzenia. Argument musi być liczbą całkowitą (np. -n 30)
- -f - określa nazwę pliku wejściowego. Argument musi być nieprzerwanym ciągiem znaków (np. -f filein.txt)
- -o - określa nazwę pliku, do którego będą zapisywane kolejne generacje. Argument musi być nieprzerwanym ciągiem znaków (np. -f fileout.txt)
- -a - określa jaki znak będzie symbolizował komórki żywe w pliku wyjściowym. Argument musi być jednym symbolem (np. -a O)
- -d - określa jaki znak będzie symbolizował komórki martwe w pliku wyjściowym. Argument musi być jednym symbolem (np. -d X)

2 Diagram modułów



Rysunek 1: Diagram modułów

3 Moduły

Program Gra w życie będzie składał się z następujących modułów:

1. **Main**

   Moduł Main będzie w pełni odpowiedzialny za sterowanie całym programem oraz obsługę flag. Moduł ten na samym początku będzie przyjmował argumenty wywołania, takie jak liczba generacji, plik wejściowy oraz plik wyjściowy. Następnie będzie deklarował dwa wskaźniki na struktury i alokował dla nich pamięć. Również będzie on odpowiedzialny za uruchamianie poszczególnych modułów w odpowiedniej kolejności;

2. **ReadFile**

   Moduł ReadFile będzie odpowiadał za pierwszą i kluczową część progamu, to znaczy za wczytanie pliku podanego przez użytkownika do programu oraz wyznaczenie rozmiaru planszy wykorzystywanej w dalszej części programu;

3. **Game**

   Moduł Game to moduł, w którym zostanie przeprowadzona pełna implementacja zasad gry w życie. Moduł Game będzie działał na zasadzie uzupełniania pomocniczej planszy na podstawie głównej planszy (struktury). Następnie zawartość uzupełnionej planszy będzie przepisywana do planszy głównej, a ona sama będzie zapełniona zerami oraz będzie czekała na ponowne zapełnienie. Będzie on zawierał również funkcję służącą do wyjęcia otaczających elementów badanego elementu. Natomiast na jej podstawie zostanie uruchomiana funkcja określająca stan tego elementu;

4. **WriteFile**

   Moduł WriteFile będzie odpowiadał za wypisywanie kolejnych generacji do pliku wyjściowego. Moduł Game będzie uruchamiał ten moduł przy każdej iteracji, a on będzie zapisywał planszę do pliku.

4 Funkcje

4.1 Main

1. int main(argc, *argv[])

   Funkcja przyjmuje dane wejściowe za pomocą standardowych argumentów wywołania argc oraz *argv[]. Odpowiada za alokowanie miejsca dla struktur oraz sterowanie programem.

4.2 Game

1. void generation(int n, grid_t * ptr, dimension_t * ptrd)

   Funckja przymuje wskaźnik na pierwszą strukturę zawierającą planszę oraz liczbę iteracji wprowadzoną przez użytkownika. Iteruje po kolejnych komórkach planszy i dla każdej z nich wywołuje funkcję *is_alive*. Następnie zapisuje stan komórki na analogicznym miejscu w kolejnej planszy

2. bool is_alive(*char cell, *grid_t) Funkcja jako argument przyjmuje wskaźnik na kolumnę i strukturę i określa stan komórki na podstawie stanów jej ośmiu sąsiadów;

3. clear(grid_t *to_clear)

   Funkcja jako argument będzie dostawała zawsze tą samą strukturę pomocniczą, która będzie zapełniana zerami (czyszczona);

4.3 ReadFile

1. void read_file(char *filename, dimension_t *dim)

   Funkcja jako argument przyjmuje wskaźnik na plik tekstowy zawierający planszę. Następnie przechodzi przez plik zliczając liczbę wierszy i kolumn. Funkcja jednocześnie sprawdza czy plik ma właściwy format. Funkcja poprzez wskaźnik nadpisuje wartości na strukturze statycznej utworzonej w Module Main.

2. void fill_in_grid(char *filename, grid_t *grid_pointer)

   Funkcja jako argumenty przyjmuje wskaźnik na plik tekstowy z planszą, oraz wskaźnik na strukturę grid_t, do której będzie zapisywana plansza z pliku.

4.4 WriteFile

1. void write_grid(FILE * out, grid_t * ptr)

   Funkcja jako argumenty przyjmuje wskaźnik na plik tekstowy do którego zapisywana będzie plansza oraz wskaźnik na strukturę przechowującą zapisywaną planszę.

2. void print_to_screen(grid_t * ptr)

   Funkcja jako argument przyjmuje wskaźnik na ostatnią strukturę. Następnie wypisuje planszę z tej struktury do konsoli.

5 Struktury

1. grid_t

   Struktura przechowująca wskaźnik na tablicę dwuwymiarową zmiennych typu *char*, opisujących stan każdej komórki. Tablica będzie zawierała bufor, czyli 2 zewnętrzne wiersze i 2 zewnętrzne kolumny wypełnione komórkami martwymi, w celu uniknięcia błędów podczas sprawdzania stanu zewnętrznych komórek planszy.

   ```
   typedef struct gird {
   char **cells;
   } grid_t;
   ```

2. dimension_t

   Struktura przechowująca wymiary planszy (liczbę wierszy i kolumn) jako zmienne typu *int*.

   ```
   typedef struct dimension {
   int rows;
   int columns;
   } dimension_t;
   ```

6 Przepływ sterowania

1. Wczytanie argumentów wywołujących program;
2. Uruchomienie modułu ReadFile;
   (a) Wczytanie rozmiaru planszy;
   (b) Wczytanie pliku z planszą do tablicy struktur;
3. Uruchomienie modułu Game;
   (a) Zapisanie planszy z pliku wejściowego do pierwszej struktury
   (b) Zapisanie kolejnej generacji do drugiej struktury na podstawie planszy z pierwszej struktury
   (c) Wywołanie modułu WriteFile
      i. Zapisanie planszy z drugiej struktury do pliku
   (d) Nadpisanie planszy w pierwszej strukturze planszą z drugiej struktury
   (e) Powtórzenie punktów 3.(b), 3.(c), 3.(d) w zależności od tego jaką liczbę generacji podał użykownik
4. Zakończenie pracy programu.

7 Główne algorytmy

Program zasadniczo opiera się na dwóch podstawowych algorytmach, które są wykorzystywane do uzupełnienia następnej planszy po każdej iteracji.

1. Zapisywanie i przechowywanie kolejnych generacji

   Program będzie korzystał z dwóch tablic (nazwijmy je X i Y) grid_t do przechowywania planszy. Wpierw plansza z pliku wejściowego, będzie zapisywana w strukturze X. Na jej podstawie zostanie wygenerowana kolejna generacja. Plansza z nowej generacji zostanie zapisana w strukturze Y i z niej wypisna do pliku. Następnie nastąpi nadpisanie planszy w strukturze X na podstawie struktury Y. Dla kolejnych generacji proces przebiega analogicznie.

2. Określanie stanu komórki w następnej generacji

   Algorytm będzie wykorzystywany przez funkcję *bool is_alive()*. Funkcja, aby sprawdzić stan sąsiadów komórki w i-tym wierszu i j-tej kolumnie dwuwymiarowej tablicy (planszy), będzie sprawdzać jaki znak

jest przypisany komórkom o indeksach [i-1][j-1], [i-1][j], [i-1][j+1], [i][j-1], [i][j+1], [i+1][j-1], [i+1][j] oraz [i+1][j+1]. Aby uniknąć sprawdzania przez funkcję indeksu poza zaalokowaną pamięcią (np. [-1][-1]) każda plansza posiada bufor, czyli dodatkowe 2 wiersze i 2 kolumny na zewnątrz składające się z komórek martwych.

8 Testy

Testy będą przeprowadzane z wykorzystaniem specjalnie przygotowanych danych wejściowych. Część danych będzie w błędnym formacie, w ich przypadku spodziewamy się odrzucenia ich przez program. Również niezbędną rzeczą do zaplanowania i przeprowadzenia testów będzie napisanie odpowiednich programów sprawdzających wybrane funkcje oraz algorytmy umieszczone w poszczególnych modułach. W każdym module wykonamy ok. 2 testów.

1. Moduł ReadFile

    - Pierwszym testem w tym module będzie wprowadzenie do programu plików tekstowych o złym formacie i zawierających niedozwolone znaki. Program powinien odrzucić te dane i napisać dlaczego są one niepoprawne(zły format pliku, niedozwolone znaki);
    - Drugim testem będzie sprawdzanie czy program poprawnie zlicza wymiary planszy we wprowadzonym pliku. Wprowadzone zostaną wygenerowane w plikach tekstowych plansze o znanych wymiarach. Program powinien za każdym razem podać poprawne wymiary;
    - Ostatnim testem w tym module będzie test polegający na sprawdzeniu, poprzez wypisanie na ekranie, poprawnego załadowania danych do tablicy dwuwymiarowej, poprzez porównanie pliku tekstowego z danymi oraz wyświetlonego wyniku;

2. Moduł Game

    - W tym teście będziemy sprawdzać czy moduł poprawnie iteruje po wektorze struktur *grid_t* zawierających plansze. W tym celu zadeklarujemy wektor tych struktur oraz zapełnimy plansze w tych strukturach w taki sposób, by układ komórek był unikalny dla każdej planszy. Program będzie iterował po wektorze struktur i przed każdą iteracją wypisywał plansze do konsoli. Jeśli program wypisze poprawnie każdą plansze w kolejności w jakiej występują w wektorze będzie to oznaczać, że moduł działa poprawnie.
    - Drugim testem w tym module będzie sprawdzenie poprawności działania algorytmu określającego stan komórki. Jako argumenty funkcji będziemy przesyłać stany otaczających komórek i zgodnie z zasadami gry będziemy oczekiwać poprawnego wyniku.

3. Moduł WriteFile

    - Aby przetestować moduł WriteFile, po perwsze sprawdzimy czy program w taki sam sposób wypisze wprowadzony i niezmieniony przez moduł Game plik tekstowy, i po drugie czy poprawnie wypisze n-tą generację planszy dla której znamy wynik.