Politechnika Warszawska
Wydział Elektryczny

SPECYFIKACJA IMPLEMENTACYJNA
PROJEKT: AUTOMAT KOMÓRKOWY

Autorzy:

Anna Głowińska
numer indeksu: 291070
anna.glowinska98@gmail.com

Adam Czajka
numer indeksu: 291063
czajka.adam147@gmail.com

Praca wykonana w ramach przedmiotu: Języki i metody programowania 2

8 kwietnia 2018

Spis treści

1 Informacje ogólne

1.1 Uruchomienie programu

Uruchomienie programu `life` następuje przez wywołanie programu `make`. Po kompilacji i uruchomieniu program pozwala użytkownikowi wybrać rodzaj sąsiedztwa, liczbę generacji do przeprowadzenia oraz dane wejściowe (generację początkową).

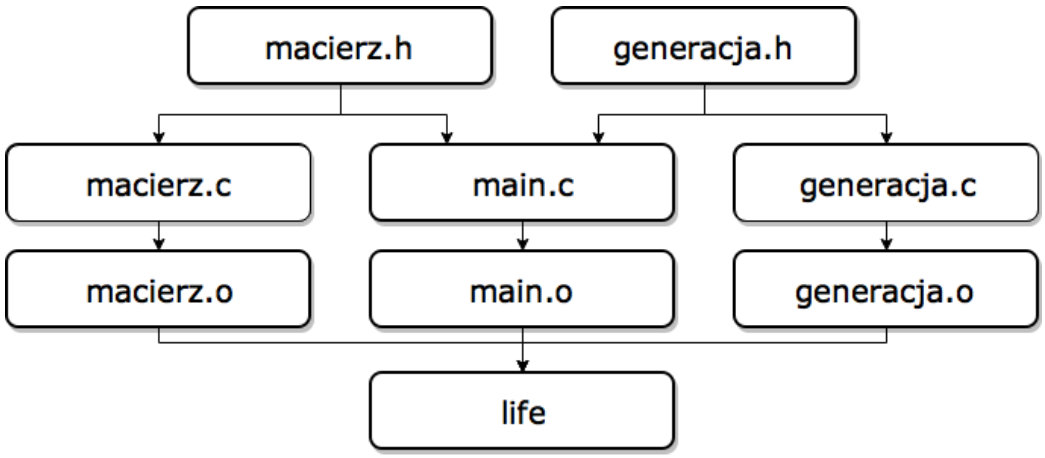
Działanie programu `life` odbywa się w oknie terminala, gdzie przebiega komunikacja między użytkownikiem a programem.

1.2 Przebieg programu

Program na podstawie pobranych informacji wykonuje zadaną liczbę generacji. Każdorazowo wypisuje obecną generację na ekran terminala i oczekuje podania znaku od użytkownika. Po wpisaniu znaku `s` następuje zapis generacji do pliku PBM, `t` - zapis generacji do pliku TXT, `q` - natychmiastowe zakończenie działania progarmu, znaki od `1` do `9` oznaczają przeprowadzenie danej liczby generacji bez zapisywania. Podanie jakiekogolwiek innego znaku jest ignorowane przez program. Wyniki (pliki PBM i TXT) są zapisywane w katalogu `life/dane/wyniki`.

2 Opis modułów

2.1 Diagram modułów



Rysunek 1: Diagram modułów.

2.2 Uzasadnienie

Diagram modułów programu `life` ma postać przedstawioną na rysunku nr 1. Moduły są między sobą powiązane relacją zależności, ponieważ wymagają ich do realizacji własnej funkcjonalności. Zależność między dwoma modułami oznaczona jest strzałką i polega na tym, że moduł w stronę którego skierowany jest grot strzałki korzysta drugiego z danych modułów - zmiana w tym module może spowodować konieczność zmiany w module oznaczonym strzałką.

2.3 Moduł generacja

Funkcja `generuj4()`

Przeprowadza generację zgodnie z zasadami „Gry w życie" z wykorzystaniem sąsiedztwa von Neumann'a. Po osiągnięciu granicy macierzy układy komórek pojawiają się po przeciwległej stronie kontynuując grę.

- funkcja jest bezargumentowa,
- funkcja jest typu `void` - nie przekazuje informacji zwrotnej.

Funkcja `generuj8()`

Przeprowadza generację zgodnie z zasadami „Gry w życie" z wykorzystaniem sąsiedztwa Moore'a. Po osiągnięciu granicy macierzy układy komórek pojawiają się po przeciwległej stronie kontynuując grę.

- funkcja jest bezargumentowa,
- funkcja jest typu `void` - nie przekazuje informacji zwrotnej.

2.4 Moduł macierz

Funkcja `inicjuj()`

Tworzy dwie macierze (dajej nazywane pierwszą i drugą macierzą) służące do przechowywania i przeprowadzania generacji oraz alokuje im miejsce w pamięci wykorzystując przeczytane z pliku wejściowego informacje o wielkości macierzy. Pierwsza macierz zawsze przechowuje obecną generację, a druga służy do przepisywania danych podczas wykoywania funkcji `generuj4()` i `generuj8()`.

- funkcja jako argument przyjmuje wskaźnik do pliku z danymi początkowymi,
- funkcja jest typu `void` - nie przekazuje informacji zwrotnej.

Funkcja `wymiary()`

Zapisuje wymiery `x` i `y` macierzy w adresach podanych przy wywołaniu. Wymiary pierwszej i drugiej macierzy są takie same.

- funkcja jako argumenty przyjmuje wskaźniki do dwóch liczb całkowitych, które są szerokością oraz długością macierzy z danymi pierwotnymi,
- funkcja jest typu `void` - nie przekazuje informacji zwrotnej.

Funkcja `wstaw()`

Wstawia wartość `w` podaną przy wywołaniu w `x`-owe (również podane przy wywołaniu) miejsce pierwszej macierzy.

- funkcja jako argumenty przyjmuje dwie liczby całkowite, z których jedna jest indeksem macierzy, gdzie będzie podstawiana druga wartość,
- funkcja jest typu `void` - nie przekazuje informacji zwrotnej.

Funkcja `wstaw2()`

Wstawia wartość `w` podaną przy wywołaniu w miejsce drugiej macierzy o współrzędnych `x y` również podanych przy wywołaniu.

- funkcja jako argumenty przyjmuje trzy liczby całkowite, z których dwie są współrzędnymi macierzy, gdzie będzie podstawiana trzecia wartość,
- funkcja jest typu `void` - nie przekazuje informacji zwrotnej.

Funkcja `wartosc()`

Zwraca wartość komórki pierwszej macierzy o podanych współrzędnych `x y`.

- funkcja jako argumenty przyjmuje dwie liczby całkowite, które są współrzędnymi macierzy,
- funkcja jest typu `int` - zwraca liczbę całkowitą.

Funkcja `wartosc2()`

Zwraca wartość `x`-owej komórki drugiej macierzy.

- funkcja jako argument przyjmuje liczbę całkowitą, która jest indeksem macierzy,
- funkcja jest typu `int` - zwraca liczbę całkowitą.

Funkcja `czytaj()`
Czyta dane z pliku podanego przy wywołaniu do pierwszej macierzy.

- funkcja jako argument przyjmuje wskaźnik do pliku z danymi początkowymi,
- funkcja jest typu `void` - nie przekazuje informacji zwrotnej.

Funkcja `wypisz()`
Wypisuje na ekran terminala obecną generację.

- funkcja jest bezargumentowa,
- funkcja jest typu `void` - nie przekazuje informacji zwrotnej.

Funkcja `rysuj()`
Zapisuje obecną generację do pliku podanego podczas wywołania. Plik jest formatu PBM.

- funkcja jako argument przyjmuje wskaźnik do pliku z danymi początkowymi,
- funkcja jest typu `void` - nie przekazuje informacji zwrotnej.

Funkcja `rysujtxt()`
Zapisuje obecną generację do pliku podanego podczas wywołania. Plik jest formatu TXT.

- funkcja jako argument przyjmuje wskaźnik do pliku z danymi początkowymi,
- funkcja jest typu `void` - nie przekazuje informacji zwrotnej.

3 Przechowywanie danych

3.1 Struktura danych

Dane wejściowe są zapisywane w strukturze znajdującej się w module `macierz`. Struktura zawiera tablicę liczb całkowitych o wymiarach zgodnych z danymi wejściowymi oraz dwie liczby całkowite `x` i `y` przechowujące dwa wymiary owej macierzy.

Dostęp do danych znajdujących się w tej strukturze następuje poprzez wywołanie odpowiednch funkcji modułu `macierz`. Ułatwia to łatwiejsze zlokalizowanie błędnych modyfikacji zawartości tej struktury.

4 Sprzęt i oprogramowanie

4.1 System operacyjny

Program `life` jest pisany, kompilowany i testowany na systemie operacyjnym `macOS High Sierra`. System ten jest z rodziny `UNIX`.

4.2 Język programowania

Program `life` został napisany w języku `C`.

4.2.1 Wersja

Program został napisany zgodnie ze standardem `ansi` oraz jest kompilowany przez program `make` z argumentam `-ansi`.

4.2.2 Kompilacja

Do tworzenia pliku wykonywalnego używany jest kompilator `gcc`. Cała procedura nastpuje poprzez wywołanie programu `make`. Przy wywołaniu programu `make` możliwe są opcje:

- uruchomienie programu (podczas którego poszczególne pliki mogą być kompilowane zgodnie z regułą pliku `Makefile`),
- sama kompilacja,
- `clean`, która powoduje usunięcie wszystkich plików o rozszerzeniu `o` z katalogu `life/src`, plików PBM z katalogu `life/dane/wyniki` oraz pliku wykonywalnego `life` z katalogu `life/bin`.

Kompilacja `gcc` następuje z argumentami `-pedantic`, `-Wall` oraz `-ansi`.

4.3 Testowanie programu

4.3.1 Testy

Do testowania używane są pliki testowe zawierające testy pojedynczych funkcjonalności programu (czytanie z pliku TXT, czytanie z pliku PBM, zapisywanie do pliku TXT etc.). Wszystkie pliki zawierające testy przechowywane są w katalogu `life/src/testy` i nazwane zgodnie z następującą koncepcją: `,,test_co-jest-testowane_co-powinno-być-wynikiem/skutkiem.c''`.

Testy pisane są na zasadzie given-when-then. Oznacza to, że kod każdego testu zostanie podzielony na trzy części:

- given - odpowiadającą za dane, które test pobiera z programu,
- when - wykonująca odpowiednie działania, wywołująca funkcje etc.,
- then - sprawdzająca poprawność wykonanych działań, wywołanych funkcji etc.

4.3.2 Debugowanie pamięci

Wycieki pamięci w programie `life` są badane i lokalizowane za pomocą narzędzia `valgrind`.

5 Wersjonowanie

5.1 Branch'e

Program jest pisany i commit'owany z jednej tylko gałęzi `master`, ponieważ w dwuosobowej zajmującej się tym programem grupie zachowane zostają zasady extreme programming zakładające obecność dwóch osób podczas pisania kodu.

5.2 Historia wersji

Historia wersji programu zostaje wypisana na ekran za pomocą komendy `$ git log`. Za pomocą opcji `amend` można cofnąć zmiany w dowolnym pliku, który był już commit'owany.