SPRAWOZDANIE - MRÓWKA LANGTONA

Karol Krukowski, Emil Stolarczyk

18.01.2024

1. Działanie mrówki Langtona

Mrówka Langtona porusza się po siatce współrzędnych, po prostokątnych polach oddzielonych od siebie liniami. Mrówka może być skierowana w jednym z 4 kierunków: *n* - północ, *e* - wschód, *s* - południe, *w* - zachód. Pola mogą być albo białe, albo czarne. W zależności od koloru pola, na którym przebywa mrówka, wykonuje ona inny ruch. Jeśli mrówka znajduje się na polu białym - obraca się o 90 stopni w prawo, zmienia kolor tego pola na czarny i przesuwa się o jedno pole w obranym kierunku. Jeśli znajduje się na polu czarnym – obraca się o 90 stopni w lewo, po czym analogicznie zmienia kolor pola na biały i przesuwa się o jedno pole. Jeśli mrówka “chce” przejść przez granicę siatki, wychodzi ona z drugiej strony wiersza lub kolumny, z której się rusza.

2. Wywołanie programu

Program przyjmuje 4 parametry koniecznych do działania programu. Są nimi: liczba kolumn siatki (-n), liczba wierszy (-m), liczba iteracji (ruchów, które mrówka ma wykonać: -i) oraz kierunek, w którym zwrócona będzie mrówka przy rozpoczęciu iteracji (-d). Jeśli parametry te nie zostaną podane, program nie zadziała i wypisze informacje o braku koniecznych parametrów oraz sposób ich podania. Istnieje również dwa opcjonalne parametry: przedrostek plików wynikowych, jeśli użytkownik chce dostać wynik działania programu w plikach, a nie na stdout (-o) oraz procent pól, które zostaną losowo zainicjowane jako pola czarne przy generacji siatki (-r).

3. Podział programu na moduły

Program podzielony jest na 5 modułów:

- ant.c - plik źródłowy, zawierający funkcje związane ze strukturą mrówki, która porusza się po siatce. Funkcja *initializeAnt* tworzy instancje struktury mrówki, ustawia jej współrzędne, tak aby mrówka znalazła się w środku siatki przy rozpoczęciu działania programu oraz pozwala wybrać kierunek, w którym zwrócona jest mrówka przy inicjalizacji ('n'/'e'/'s'/'w'). Na koniec, funkcja zwraca instancję mrówki o danych parametrach. Funkcja *move* jest odpowiedzialna za logikę ruchów mrówki po siatce. Jako argumenty przyjmuje ona wskaźnik na mrówkę, zainicjowaną w działaniu programu, oraz komórkę siatki, w której się ta mrówka znajduje. Najpierw, funkcja sprawdza, czy komórka, w której mrówka się znajduje jest biała. Jeśli jest, zmienia kolor tej komórki na czarny (*false* odpowiada kolorowi czarnemu, *true* - białemu), zmienia kierunek mrówki oraz przesuwa ją o jedno pole w zależności od kierunku, w którym zwrócona jest mrówka.
- ant.h - plik nagłówkowy, przekazujący funkcje pliku ant.c oraz deklarujący strukturę *ant*, symbolizującą mrówkę Langtona. Posiada ona zmienne n - współrzędną kolumny, w której znajduje się mrówka, m - współrzędną wiersza, oraz direction – kierunku, w którym zwrócona jest mrówka. Zmienna ta przechowuje dane typu char i działa dla liter n, e, s, w, odpowiadającym kierunkom geograficznym.
- grid.c - plik źródłowy, zawierający funkcje związane z generacją i drukowaniem siatki, po której porusza się mrówka. Funkcja *initializeGrid* inicjalizuje siatkę współrzędnych, przyjmując wskaźnik na macierz, do której zapisywane będą wartości typu bool, odpowiadające kolorowi komórek siatki *(true* - biały, *false* – czarny)*,* oraz wymiary n x m siatki. Funkcja *print*Grid drukuje wizualizację siatki i mrówki, iterując przez każde położenie mrówki w jej ruchu. Funkcja *writeToFile* działa jak funkcja *printGrid*, ale zamiast wypisywania iteracji ruchu na stdout, zapisuje ona każdą iteracje do nowego pliku wyjściowego, nazywanego <*przedrostek podany przez użytkownika_nr.iteracji*> oraz pliku bez numeru iteracji, zapisujący wygląd siatki przed rozpoczęciem ruchu mrówki. Funkcja *randomizeGrid*, odpowiada za zapełnienie części siatki czarnymi polami, określonej przez użytkownika procentem pól które mają być czarne przy generacji pola.
- grid.h - plik nagłówkowy, przekazujący funkcje pliku grid.c do main.c
- main.c - plik wykonujący działanie programu. Pobiera on argumenty wywołania i przy użyciu funkcji getopt przetwarza je na argumenty do wywołania funkcji z plików źródłowych. Po przetworzeniu argumentów wejściowych, funkcja main przeprowadza obsługę błędów: sprawdza czy obowiązkowe argumenty wywołania programu zostały podane oraz czy zostały podane w prawidłowy sposób. Jeśli nie zostały, zwraca adekwatną informację na stdout. Po przetworzeniu i sprawdzeniu argumentów, funkcja inicjalizuje macierz komórek, mrówkę oraz siatkę współrzędnych. Następnie drukuje wizualizację iteracji mrówki i jeśli zostały podane odpowiednie parametry – zapisuje każdą iterację do plików wyjściowych zamiast na stdout lub/i zapełnia określoną przez użytkownika część siatki polami czarnymi.

4. Przykładowe działanie programu

- Wywołując: "*./a.out -n 5 -m 6 -i 10 -o przyklad -d n"* – program utworzy siatkę z 5 kolumnami i 6 wierszami oraz zapisze każdą z iteracji do plików wyjściowych: *przyklad* (siatka przed rozpoczęciem iteracji), *przyklad_1* ... *przyklad_10* oraz ustawi kierunek mrówki na "n" (skierowana będzie do góry)
- "*./a.out -n -2 -m 6 -i 10 -o przyklad -d n"* – nastąpi błąd programu, i jego działanie nie zajdzie. Na stdout zostanie wypisana informacja, że parametr –n musi być większy niż 0
- "*./a.out -n 5 -m 6 -i 0 -o przyklad -d x"* – błąd programu, informacja o błędzie, parametr kierunku mrówki -d, może być tylko char-em: 'n'/'e'/'s'/'w'
- "*./a.out -n 5 -m 6 -i 10 -d n"* – program utworzy siatkę z 5 kolumnami i 6 wierszami, wydrukuje na stdout 10 iteracji (10 "ruchów" mrówki) oraz ustawi kierunek mrówki na "n" (skierowana będzie do góry)
- "*./a.out -n 5 -m 6 -i 10 -d n –r 50"* – program utworzy siatkę z 5 kolumnami i 6 wierszami, wydrukuje na stdout 10 iteracji (10 "ruchów" mrówki), ustawi kierunek mrówki na "n" (skierowana będzie do góry) oraz ustawi 50% pól siatki na pola czarne przy jej generacji

5. Wnioski

Pisanie tego programu uświadomiło nam, że poczyniliśmy postępy w rozumieniu i metodykach języka C, względem początku semestru. Nie natrafialiśmy na niezrozumiałe "segmentation fault-y" (co zdarzało się przy pisaniu poprzednich zadań) i ogólnie mówiąc, rozumieliśmy, jak musi wyglądać nasz kod, aby spełniać postawione założenia. Z przeprowadzonych przez nas testów, program działa poprawnie, zarówno jak i obsługa błędów.