Kamil Lipski

Dokumentacja projektu

PRI - zadanie laboratoryjne 3

Warszawa, 2017

1. Cele

Jest to program zarządzający strukturami dynamicznymi (stosem z kolejkami FIFO). Wyobraźmy sobie taką sytuację: niech istnieje sobie segmentowa wieża (nowy segment można dokładać na górze). Wieża ma szyb prowadzący w dół do którego wrzuca się kulki o losowej barwie i kolejnym numerze. Kulka wrzucona w szyb z szansą p przeleci na niższy poziom, a z szansą 1 – p zostanie złapana przez wbudowaną w segment zaślepkę i skierowana do rurki na kulki wychodzące. Na końcu tej rurki można odebrać po jednej kulce z utworzonej kolejki i wrzucić od góry ponownie. Albo zniszczyć. Kulki w ostatnim segmencie zawsze kierowane są do rurki wyjściowej. Program pozwala w dowolnym momencie w pełni podglądać i modyfikować struktury danych. Dodatkowo w programie został zaimplementowany zapis i odczyt dynamicznej struktury z dysku (binarny). Podanie przy uruchamianiu programu nazwy pliku z zapisem wczytuje na starcie dany plik.

2. Opis działania programu

Zmienne: kolejka - ilość elementów w kolejce, i - kolejny numer kulki, rozmiar - rozmiar stosu, ilosc_poz - zmienna pomocnicza (używana przy symulacji przejścia kulki przez wieżę), los_prawd - zmienna pomocnicza służąca do określania, czy kulka przejdzie przez dany segment wieży,

czy nie, k, komenda - zmienne służące do prawidłowego działania programu, warunek - zmienna służąca do sprawdzania poprawności danych wprowadzonych przez użytkownika, petla1, petla2 - zmienne służące m.in. do obsługi parametrów uruchomienia, decyzja - zmienna przechowująca decyzję użytkownika, numer, kolor, prawd - zmienne użyte w strukturach opisujące parametry kulki\stosu

W programie wykorzystuję również zmienne wskaźnikowe, służące do obsługi kolejki i stosu

Tablice: nazwa[50] - przechowuje nazwę pliku do odczytu (podawaną przez użytkownika)

Struktury: element 2, kolejka - struktury służące do implementacji kolejki, element, stos - struktury służące do implementacji stosu

Użyte funkcje:

- struct kolejka *Insert(struct kolejka *tail, struct element2 data2) - umieszcza nowy element w kolejce,
- struct kolejka *Remove(struct kolejka *head) - usuwa element z kolejki,
- struct stos *push(struct stos *top, struct element data) - umieszcza na stosie nowy element,
- struct stos *pop(struct stos *top) - zdejmuje element ze stosu

Użytkownik decyduje, czy chce wczytać dane z pliku. Jeśli tak, to musi podać jego nazwę. Dalej program prosi

użytkownika o podanie ilości segmentów, które ma dołożyć do wieży i ilości nowych kulek, które mają być do niej wrzucone. Teraz użytkownik ma do wyboru: dołożenie do wieży nowego segmentu, usunięcie segmentu z wieży, wrzucenie nowej kulki (program symuluje jej przelot przez wieżę i podaje, gdzie się zatrzymała; jeśli spadła na poziom 0, to została zniszczona, jeśli nie, to została dodana do kolejki), ponowne wrzucenie kulki do wieży, zniszczenie kulki (obydwie komendy dotyczą pierwszej kulki znajdującej się w kolejce) oraz wyświetlenie danych (ilość poziomów wieży i kulki znajdujące się w kolejce). Jeśli użytkownik będzie chciał wyjść z programu, zostanie zapytany, czy chce zapisać dane do pliku. Jeśli tak, to dane zostaną zapisane do pliku o nazwie: *dane.bin*.

3. Założenia

- kolejka i stos są zaimplementowane dynamicznie,
- prawdopodobieństwo przejścia przez dany segment jest unikalne (zostaje wylosowane za pomocą funkcji rand()),
- program został podzielony na pliki: *main.c*, *funkcje.c*, *funkcje.h*,
- w przypadku błędu otwarcia pliku użytkownik zostanie poproszony o podanie nazwy innego pliku do odczytu,

- w przypadku podania przez użytkownika błędnych komend, program wypisze komunikaty typu: *Blad! Jeszcze raz!,* itp.,
- kulki mogą mieć pięć różnych kolorów: biały, czarny, niebieski, zielony, czerwony,
- w przypadku próby usunięcia kulek, kiedy kolejka jest pusta, wrzucenia kulki do wieży, gdy stos jest pusty itd., program wypisze odpowiedni komunikat,
- jeśli próba otwarcia pliku do zapisu danych zakończy się błędem, program zakończy swoje działanie.

4. Przykładowe testy

Sprawdzenie poprawności odczytu/zapisu danych z/do pliku, sprawdzenie poprawności działania wszystkich elementów programu (poprawności wykonywania komend i reakcji na różne sytuacje).

5. Schemat blokowy