MÄSTARPROV 9

Tillämpning av modul 1-9

Game of Life – Del 2

Introduktion

I mästarprov 8 implementerades en enkel version av Conway's Game of Life. I denna uppgift ska den implementationen byggas vidare på.

När man vill att ett program ska använda sig av en viss data finns det flera tillvägagångssätt för att uppnå det. Man kan låta användaren skriva in data medans programmet exekverar, vilket gjordes i mästarprov 5 och 8 (`load_custom()`). Ett alternativ, som användes i mästarprov 8, är att ha funktioner som lägger in data. En nackdel med det är att man måste kompilera om programmet varje gång man vill fylla på med data. I denna uppgift ska vi använda oss av ett tredje sätt som är mer vanligt när man hanterar större mängder av data, data ska läsas och skrivas till filer. Med denna lösning uppstår även ett behov av att dynamiskt allokera minne för den data som läses in, även detta ska användas i denna uppgift.

När källkoden till ett program blir omfattande är det mycket opraktiskt att ha all kod i samma källkodsfil. Det man istället gör är att dela upp koden till programmet i flera olika filer, det är viktigt att denna uppdelning är logisk och konsekvent. I denna uppgift ska källkoden delas upp i fem olika filer.

Uppgiften ska lösas enskilt.

Syfte

Syftet med uppgiften är att testa att du kan applicera den kunskap du bemästrat i Modul 1-9 och sätta samman ett komplett program i enlighet med givna instruktioner.

Uppgift

Din uppgift är att ändra i din implementation från mästarprov 8 så att implementationen läser in data till strukturen `field` från en fil och att vid avslut av exekveringen av programmet så ska den aktuella konfigurationen skrivas till en fil. Namnen på filerna ska ges som parametrar till programmet. Vidare ska din implementation delas upp i flera filer.

Följande förändringar ska göras (räknas ej upp i den ordning de ska genomföras, välj en lämplig ordning):

- Ta bort funktionerna `load_random`, `load_glider`, `load_semaphore` och `load_custom`
- Skapa en struktur som samlar data kopplat till ett spel (följande struktur ska användas):

```
typedef struct {
      int rows;
      int cols;
      cell **cells;
} field;
```

- Implementera funktionerna `check_prog_params`, `load_config_from_file` och `save_config_to_file`
- Implementera en funktion för att dynamiskt allokera minne för den data som läses in från fil och en funktion för att avallokera det dynamiskt allokerade minnet
- Uppdatera de funktioner som påverkas av ovanstående förändring
- Dela upp programmet i fem källkodsfiler:
    - `mp9.c` – Ska innehålla funktionerna `main` och `check_prog_params`
    - `game_of_life.h` – Ska innehålla deklarationer av strukturer och de funktioner som är kvar från implementationen till mästarprov 8 förutom `main`-funktionen. Ska även innehålla deklarationerna för de två nya funktionerna kopplat till dynamisk minneshantering
    - `game_of_life.c` – Ska innehålla definitioner för funktionerna i `game_of_life.h`
    - `game_of_life_file_handler.h` – Ska innehålla deklarationer för de två nya funktionerna `load_config_from_file` och `save_config_to_file`
    - `game_of_life_file_handler.c` – Ska innehålla definitioner för funktionerna i `game_of_life_file_handler.h`

För att se hur ditt program ska fungera finns det ett körbart komplett program som du får undersöka tillsammans med denna specifikation. Det körbara programmet nås via länk i beskrivningen till detta mästarprov.

Specifikation för de tre nya funktionerna samt dataformatet i konfigurationsfiler

Följande beskriver gränsytan för de nya funktionerna:

```
/* Description: Checks the parameters to the program. Checks if the
 *              call to the program has the right number of
 *              parameters. Open the input and output files.
 * Input:       The parameters to the program and two pointers to file
 *              pointers.
 * Output:      Returns 0 when the files are correctly opened.
 *              Returns a non-zero value on failure.
 */
int check_prog_params(int argc, const char *argv[],
                      FILE **in_file_p, FILE **out_file_p);

/* Description: Loads a configuration to the field structure from a
 *              file. It is the responsiblity of the caller to
 *              deallocate the dynamically allocated memory in the field
 *              structure through a call to the function destroy_field.
 *              The file pointed to by fp is closed.
 * Input:       A pointer to where the created field structure should be
 *              assigned and a file pointer to the file with the initial
 *              configuration.
 * Output:      Returns 0 on success, the field structure is created
 *              with the configuration from the file.
 *              Returns a non-zero value on failure.
 */
int load_config_from_file(field *the_field, FILE *fp);
```

```
/* Description: Saves the current configuration of the field to a
 *              specified file. The file pointed to by fp is closed.
 * Input:       The field structure and a file pointer to the file
 *              where the final configuration should be saved.
 * Output:      Returns 0 on success, the current configuration in the
 *              field is written to the file.
 *              Returns a non-zero value on failure.
 */
int save_config_to_file(const field the_field, FILE *fp);
```

Följande är ett exempel på dataformatet i en konfigurationsfil (inga tomma rader innan data):

```
5,5
00000
00*00
00*00
00*00
00000
```

Den första 5:an anger antal rader och den andra 5:an antal kolumner. Tecknet `*` representerar en levande cell och tecknet `0` representerar en död cell.

Tre givna konfigurationsfiler (`glider.gol`, `gun.gol`, `semaphore.gol`) finns till din hjälp, de nås via beskrivningen till detta mästarprov.

Krav på implementationen

Här nedan finns det ett antal krav som programmet ska uppfylla:

- Algoritmen i mästarprov 8 ska följas, förutom nödvändig och logisk uppdatering enligt denna uppgift.
- De givna funktionsdefinitionerna för `check_prog_params`, `load_config_from_file` och `save_config_to_file` ska ej ändras.
- Den givna strukturen `field` ska användas och får ej ändras.
- All utskrift på skärm ska se exakt ut som i det tillhandahållna körbara programmet och som visas vid testkörningar vid inlämning.
- Källkodsfilerna ska heta `mp9.c`, `game_of_life.h`, `game_of_life.c`, `game_of_life_file_handler.h`, samt `game_of_life_file_handler.c`.
- I den beskrivande texten i filerna ska du inkludera ditt namn och din cs-användare.
- Programmet ska kunna kompileras med kompilatorn `gcc` med flaggorna `-Wall` och `-std=c99`.

Programmet behöver bara kontrollera att viss indata är korrekt. Följande ska kontrolleras:

- Att filer går att öppna
- Att första raden i en konfigurationsfil är korrekt formaterad

Vid något av dessa fel ska felutskriften ske till `stderr`. För övrigt krävs ingen validering av indata, programmet ska gå att använda på det sätt som illustreras i det tillhandahållna körbara programmet. Om du väljer att validera data ska valideringen vara korrekt.

Redovisning

Uppgiften redovisas genom att lämna in källkodsfilen via webbgränssnittet i Labres, se länk i beskrivningen till detta mästarprov. Din inlämnade fil kompileras och testkörs. Du kan lämna in flera gånger.

När du vill få din inlämning bedömd, gör mästarprovet för denna modul (MP9 - Mästarprov modul 9: Önskan om bedömning). I det mästarprovet ska du bara skriva din cs-användare och lämna in provet.

Tips

Här följer några tips som kan hjälpa till:

- Börja med att förstå uppgiften: Vad ska göras? Vilka krav finns det? Vad ska lämnas in?
- Ta fram en lösning på vad du vill uppnå innan du börjar implementera.
- Utveckla programmet stegvis.
- Försök att testa varje steg utförligt innan nästa steg tas och kom ihåg att testa att tidigare steg fortfarande fungerar när nästa steg testas.
- Funktionerna `fscanf` och `fprintf` kan vara till användning. Kolla noga upp hur de fungerar.

Kvalitetskriterier

Den inlämnade lösningen kommer att bedömas enligt följande kriterier:

Kriterium	**Godkänd**	**Godkänd med anmärkning**	**Ofullständig**
Kompilering	Utan varning	Mindre allvarlig	Allvarlig
Testkörning/ Korrekthet	Utan fel	Mindre fel	Felaktig output
Läsbarhet (kommentarer, indentering, variabel-deklaration, namngivning)	Bra kommentarer Korrekt indentering Konsekvent variabeldeklaration Bra och konsekvent namngivning	Mindre brister i kommentarer Något fel i indentering Något fel i samband med variabeldeklaration Mindre problem med namn-givning	Stora brister i kommentarer Dåligt indentering Problem med variabeldeklaration Bristfällig namngivning
Valstrukturer	Bra struktur Bra val av villkor	Villkorsoperatorn används Mindre bra val av villkor	Allt på en rad Felaktiga villkor
Loop-strukturer	Bra val av loop-konstruktion Bra val av villkor	Mindre bra val av loop-konstruktion Mindre bra val av villkor	Felaktiga villkor `break` och `continue` används i onödan
Funktioner	Programmet är uppdelat i lämpliga funktioner med bra parametrar och returvärden	Mindre problem med val av parametrar/returvärden Någon funktion utför för många orelaterade uppgifter	Funktioner saknas Många funktioner utför för många orelaterade uppgifter Felaktiga val av parametrar/returvärden
Arrayer	Arrayer används på ett korrekt sätt	Mindre bra användning av array	Indexeringsfel Indexerar utanför array Felaktig användning av array
Pekare	Används korrekt	Mindre problem	Felaktig användning
Egen-definierade datastrukturer	Används korrekt Används på ett konsekvent sätt	Någon miss i användandet	Saknas Används felaktigt Används inkonsekvent
Filhantering	Korrekt	Glömmer i enstaka fall `fclose()` vid felhantering Mindre miss vid felhantering	`fclose()` saknas i det normala exekveringsflödet `fclose()` saknas vid felhantering
Minnes-hantering	Korrekt	Glömmer i enstaka fall frigöra minne vid felhantering Mindre miss vid felhantering	`free()` saknas i det normala exekveringsflödet `free()` saknas vid felhantering
Program-struktur	Måsvingar placeras konsekvent Måsvingar används till alla val- och loop-konstruktioner Funktionsdeklarationer och funktionsdefinitioner är konsekvent placerade Programmet är korrekt uppdelat i olika filer	Fall av inkonsekvent placering av måsvingar Inkonsekvent placering av funktionsdeklarationer och funktionsdefinitioner Mindre problem med programmets uppdelning i olika filer	Måsvingar saknas eller placeras hur som helst Onödigt många return-satser i en funktion `exit()` används Annat än 0 returneras i main Globala variabler Kommandona `goto` och/eller `longjmp` används Bristfällig uppdelning av programmet i olika filer
Algoritm	Följer given algoritm	Mindre avvikelse från algoritm	Följer ej algoritm Ändrar given kod som ej ska ändras