Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Semestrální práce

Generátor konečného automatu

Plzeň 4. ledna 2021 Jan Pelikán

1 Zadání

Naprogramujte v ANSI C přenositelnou konzolovou aplikaci, která načte ze souboru definici konečného automatu, který přijme textový řetězec popsaný regulárním výrazem. A vygeneruje validní zdrojový kód v ANSI C, který bude představovat implementaci daného konečného automatu.

Program se bude spouštět příkazem: **fsmgen.exe <file>**. Symbol `<file>` zastupuje jméno souboru s popisem konečného stavového automatu. Váš program tedy může být během testování spuštěn například takto:

...\>fsmgen.exe graph.gv

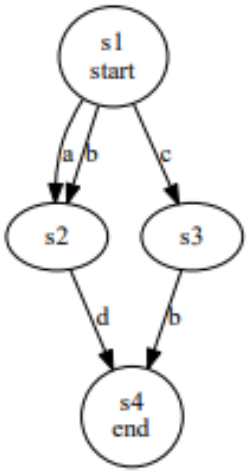
Výstupem bude validní zdrojový kód v ANSI C, představující implementaci daného automatu. Pokud nebude uveden právě jeden argument, vypíše chybové hlášení a stručný návod k použití programu v angličtině podle běžných zvyklostí. **Vstupem programu jsou pouze argumenty na příkazové řádce, interakce s uživatelem pomocí klávesnice či myši v průběhu práce programu se neočekává**.

Obsah

2 Analýza úlohy

Jako hlavní problém, který zadání přináší bych nazval naprogramováním konzolové aplikace. Tento problém se dá dekomponovat do menších problémů, jako například: čtení ze souboru, generování kódu či implementace vygenerovaného kódu.
Začněme tedy čtením ze souboru, to lze realizovat pomocí funkcí *fgetc()* nebo *fgets()*. Použitím první zmíněné funkce budeme číst znak po znaku a použitím druhé funkce lze číst celou řádku až v souboru. Jak lze vidět na ukázce níže, na každé řádce se nachází pouze jedna podstatná informace, tudíž jsem pro svoji implementaci zvolil druhou možnost (*fgets()*). Soubor má vždy stejnou strukturu, tudíž není problém se čtením různých souborů a obecné předpoklady jsou vždy splněny. Nastává ale další problém, kte-

```
digraph G
{
⟶//nodes
⟶s1[label=start];
⟶s2;
⟶s3;
⟶s4[label=end];
⟶//edges
⟶s1->s2[label=a];
⟶s1->s3[label=c];
⟶s1->s2[label=b];
⟶s3->s4[label=b];
⟶s2->s4[label=d];
}
```



Obrázek 2.1: Ukázka ze zadaní

rým je získání pouze podstatných informací a oddělení je od zbytku řádky. Důležité informace pro mě jsou: název vrcholu (s1), a pokud má, tak jeho label (start), u hrany jsou důležité informace z jakého a do jakého vrcholu vede a také její label. Jelikož budeme načítat celou řádku, přicházejí v úvahu již implementované funkce pro práci se "stringem". Jedná se o funkce *str...*, které se mohou využít pro rozdělení řádku do příslušných tokenů nebo vyhodnocení, zda se v zadaném řetězci nachází nějaký hledaný řetězec.
Dále bych zkusil analyzovat problém implementace. Tohle byl pro mě dlouho

neřešitelný problém. Nevěděl jsem, jak načtená data uchopit, a jak celý soubor vytvořit. První problém byl vymyšlení vlastní struktury. Možným řešením bylo udělat strukturu grafu, která by uchovávala všechny informace o vrcholech a hranách. Určitě je toto řešení realizovatelné, ale mě se nepovedlo. Místo toho mě napadlo vyřešit problém pomocí dvou struktur, které na sebe odkazují. Další výzvou bylo udělat funkce, které by celý graf zprovoznili. Jednou možností bylo mít jeden generovaný soubor, ve kterém budou funkce pro správu grafu a na konci bude funkce main. Při programování jsem náhodou narazil i na další způsob řešení implementace. Funkce budou v jiném souboru a při pozdějším generování souboru ušetřím drobet času. Využil jsem tedy konstrukci **#include <file>**.
Posledním problémem, který jsem ještě nezmiňoval, bylo generování souboru. K zápisu do souboru lze využít funkci *fputs()*, která zapisuje do souboru. Napadají mě dvě možnosti řešení. Jednou z možností je vytvoření si jednoho velkého bufferu (pole znaků) a zapisování do něj a druhou možností je postupné zapisování do souboru (využití více metod fputs). Zvolil jsem druhou možnost, protože jsem se vždy postupně "zbavoval"potřebného textu k vygenerování.

3 Popis implementace

Jako první budu popisovat implementaci generátoru. Generátor je implementován jako jeden soubor **fsmgen.c**. Nejdříve jsem implementoval funkce, které později využívám v mainu. První funkcí je `odstran_tab(char* vstup)`. Tato funkce se využívá při generování souboru fsm.c a slouží pro odstranění tabulátoru nebo mezer ze začátku načtené řádky (viz "—>"na obrázku 2.1). Další dvě funkce (`introInfo() a inputInfo()`) jsou pouze pro vypisování informací pro uživatele. Zajímavější funkce už jsou `pridej_vrchol(char[] radka) a pridej_hranu(char[] radka)`. Obě funkce fungují na podobném principu. Rozdílné jsou jen v tom, zda v zadaném parametru najdeme informace o vrcholu či hraně. Funkce potom řádku ořezají abych dostal pouze podstatné informace. Využívám funkce `strtok(char *str, const char *delim)`, která rozděluje vstup podle zadaného rozdělovače. Toho využívám k získání názvů vrcholů a labelů. Funkce vrací předpřipravený řetězec, který je připravený pro zápis do generovaného souboru (validní ANSI C kód). Poslední funkcí v souboru je funkce main. Zde zkontroluji vstupní parametry a buďto program pokračuje se zadaným vstupem, nebo se program ukončí a vypíše hlášku v angličtině uživateli. Poté se otevřou soubory a začne čtení souboru. Jako první se přečtou 3 řádky ze souboru, to jsou řádky, které neobsahují žádné důležité informace. Poté začne while cyklus přes všechny řádky souboru, pokud je na zadané řádce *//edges* tak program skočí do vnitřního cyklu, kde přidává hrany. Pokud se narazí na konec souboru z obou while cyklů se vyskočí. Implicitně se do generovaného souboru jako první přidávají vrcholy a poté až hrany. Při dokončení čtení ze souboru dopíšu do generovaného souboru konec a zavřu všechny soubory. Nakonec vypíšu hlášku o úspěšném provedení.
Jako další krok bych popsal implementaci generovaného souboru. Soubor jako takový obsahuje pouze jednu funkci a to je funkce main. Další funkce se přidávají pomocí **#include** z dvou dalších souborů. Na začátku funkce pouze zkontroluji vstupní parametry. Pokud je zadán pouze jeden parametr, proběhne zbytek programu, to znamená, přidají se všechny vrcholy a hrany do seznamu vrcholů. Po přidání všech hran a vrcholů se zkontroluje vstupní řetězec a uživatel se informuje o správnosti/nesprávnosti jeho vstupu. Soubor buďto skončí jako *EXIT_SUCCES* nebo *EXIT_FAILURE* a vždy se z paměti vyčistí všechny alokované rámce paměti. Funkce, které tento soubor, využívá jsou v souborech *fsm_sprava.c* a *fsm_pridani.c*. Tyto soubory "includuji"hned na začátku souboru.

První soubor, který bych popsal, je *fsmgen.h*, v tomto souboru deklaruji svoje dvě struktury. Strukturu hrana a strukturu vrchol. Podíváme se na tyto struktury blíže. Struktura vrchol obsahuje jméno a label vrcholu, počet hran, které jsou k vrcholu připojeny a dále odkazuje na následující vrchol (takový spojový seznam) a pole hran (odkaz na strukturu hrany). Struktura hrany obsahuje názvy obou svých vrcholů, svůj label a také odkaz na další vrchol neboli vrchol do kterého hrana vede.

```
15      /**...*/
18      typedef struct struct_vrchol {
19          char *jmeno;
20          char *label;
21          int pocet_hran_vrcholu;
22          struct struct_vrchol *dalsi_vrchol;
23          struct struct_hrana **hrany;
24      } vrchol;
25
26      /**...*/
29      typedef struct struct_hrana {
30          char *z_vrcholu;
31          char *do_vrcholu;
32          char label;
33          struct struct_vrchol *dalsi_vrchol;
34      } hrana;
35
```

Obrázek 3.1: Ukázka použitých struktur

Dalším v pořadí je soubor *fsm_sprava.c*. V tomto souboru řeším uvolňování paměti a kontrolu vstupu. Funkce `free_vsechno(vrchol* vrcholy)` není ničím zvláštní. Jako parametr předávám pole vrcholů, které postupně čistím. Nejdříve uvolním všechny hrany, potom uvolním pole hran a nakonec i celý vrchol. Využívám zde spojového seznamu v posunu na další vrchol. Funkce `zkouska_grafu(vrchol* vrchol)` kontroluje, zda se v zadaném seznamu nachází vrchol, který by měl label start. Pokud label start najde, vrací 1 (true). Této funkce využívám ve funkci `zkouska_vstupniho_retezce(char`

`*retezec, vrchol *vrchol)`, tato funkce prochází celý řetězec a porovnává znak po znaku s labelem každé hrany a pokud se rovnají, posunuje se na další vrchol dané hrany. Pokud se shodná hrana nenajde, program končí chybou a vypsáním chybového hlášení. Pokud celý řetězec projde úspěšně, tak se na konci vyhodnotí, zda koncový bod má label "end"nebo ne. Tudíž jestli jsme došli do cíle nebo ne.
Posledním souborem je *fsm_pridani.c*, v tomto souboru jsou funkce pro přidání hrany a vrcholu. Funkce pro přidání vrcholu v parametrech obdrží seznam již vytvořených vrcholů, svůj budoucí název a label. Na začátku si alokujeme pamět pro daný vrchol, naplním počátečními hodnotami, poté alokuji pamět ještě pro pole hran daného vrcholu. Poté jen procházím seznam a hledám poslední vrchol, když najdu poslední vrchol seznamu, tak přiřadím nově vytvořený vrchol na konec. Funkce pro přidání hrany funguje obdobně ale s menšími úpravami. Po přiřazení vstupních parametrů do struktury začnu prohledávat seznam vrcholů a hledám vrchol, ze kterého hrana vychází. Když najdu, zvýším vrcholu počet hran o 1 a realokuji pamět, poté jen vrcholu přiřadím hranu. Poté už jen v seznamu najdu do kterého vrcholu hrana vede a to je "dalsi_vrchol"té hrany.

4 Uživatelská příručka

Samotný program pro generování se podle zadání spustí příkazem **make**. Tento příkaz vytvoří spustitelný soubor **fsmgen.exe**. Dále už lze spustit program fsmgen s jedním povinným parametrem a tím je název souboru s daty (definicí grafu). Pokud proběhne všechno v pořádku, tak se vygeneruje soubor fsm.c. Tento soubor už si bohužel budete muset ručně přeložit pomocí vašeho překladače. Například příkazem

```
gcc fsm.c -o fsm
```

A poté spustit program fsm.exe s jedním argumentem, kterým je cesta po hranách grafu. Program poté vypíše, zda zadaný řetězec úspěšně prošel grafem či ne.

5 Závěr

Musím říci, že před začátkem práce na semestrální práci jsem byl celým jazykem C zmaten a nedokázal si představit, že dokážu něco naprogramovat. Většina práce byla vymýšlení toho, kam mám přidat a naopak kde ubrat * neboli pointer. (Těším se na zkoušku, která mi ukáže, jestli jsem se něco naučil nebo bez stackoverflow nedám ani ránu.) Celá semestrální práce mi dost pomohla v tom uvědomit si, jak by asi programování v ANSI C mohlo vypadat. A ukázalo mi, že jazyk C asi nebude ta správná cesta. Ale celou práci bych hodnotil kladně. Semestrální práce splnila moje očekávání na výbornou. Se svojí prací jsem spokojen i když určitě není ideálně řešená a mohla by fungovat lépe. Myslím si, že zadání jsem splnil správně.