w październiku 2016

Zdzisław Płoski

Struktura programów w języku C. Moduły i funkcje

Zadanie 3 na pracownię C
IIUWr, 2016

Z dużymi programami jest jak z dużymi książkami: trzeba jakoś zapanować nad ich rozmiarem. Dzieli się je na mniejsze jednostki, te z kolei na jeszcze mniejsze, aby ogarnąć ich (inżynierską, konstrukcyjną) złożoność. W książkach mamy części, rozdziały, podrozdziały itd. W programach – moduły (w języku C oddzielnie kompilowalne, w tym i tylko w tym sensie traktowane jako kompletne części programu), definicje procedur i funkcji, bloki instrukcji.

Znasz zapewne 1000-stronicowe dzieło Cormena i in. o skromnym tytule *Wprowadzenie do algorytmów* [1]. Z techniczną złożonością jego treści poradzono sobie w opisany sposób. Wszakże nie trzeba opasłego tomu, aby z pożytkiem zastosować metodę strukturalizacji tekstu. W małej książeczce Cormena *Algorytmy bez tajemnic* [2] zauważysz podobne zasady składu. Przejdźmy jednak do programowania.

W tym zadaniu masz przećwiczyć modularyzację i strukturę programów w języku C, a więc podział programu na pliki i funkcje. Aby się do tego odnieść, posłużymy się gotowym rozwiązaniem algorytmicznego problemu sortowania przez wyliczanie, opisanym w [2]. Ponieważ program ma być testowany przez automatyczną sprawdzarkę zamontowaną w środowisku Moodle, akceptujującą tylko jeden plik wejściowy, **większy ciężar sprawdzania tego zadania spocznie na Tobie i Twoich Opiekunach na pracowni C.**

Przystępując do rzeczy: napisz trójmodułowy program sortujący wartości n-elementowej tablicy A, której elementy, nazywane kluczami, należą do przedziału od 0 do m liczb całkowitych ($m \leqslant n$). Zastosuj metodę opisaną szczegółowo na stronach *Algorytmów bez tajemnic* (książkę znajdziesz np. Bibliotece Wydziałowej WMiIUWr). Dla ułatwienia w dodatku 1 na końcu tego tekstu cytujemy wszystkie niezbędne części realizacji algorytmu wyrażonej w pseudokodzie.

Masz przygotować do osobnej kompilacji i wspólnej konsolidacji następujące pliki:

`spz_main.c` – plik główny zawierający wywołanie funkcji wprowadzania i sprawdzania danych, wywołanie funkcji sortowania i wywołanie funkcji wyprowadzania wyników;

`spz_io.c` – plik zawierający definicje dwóch funkcji:

a) wprowadzania i sprawdzania poprawności danych wejściowych;

b) wyprowadzania wyników;

`spz_alg.c` – plik z definicjami funkcji składających się na implementację algorytmu sortowania przez zliczanie (zob. pseudokod w dodatku 1).

Ponadto w pliku nagłówkowym `f-protot.h` umieść prototypy wszystkich funkcji składowych programu i definicję maksymalnego rozmiaru sortowanych danych (powiedzmy 10000).

Funkcja wprowadzania i kontroli poprawności danych ma wykrywać i sygnalizować trzy rodzaje błędów (tekst sygnałów musi być literalnie taki sam jak niżej, ze względu na „IQ" sprawdzarki. Podajemy go w postaci konkatenacji napisów C:

1. `"\nError: size of input table out of range;\n"`
   `" must be between 1 and %d (inclusive).\n"`

2. `"\nError: cardinality of key set out of range;\n"`
   `" must be between 1 and %d (inclusive).\n"`

3. `"\nError: value of element of input data out of range;\n"`
   `" must be between 0 and %d (inclusive).\n"`

Ogółem masz do zdefiniowania (przepisania z pseudokodu na język C) następujące jednostki danych globalnych i funkcje.

Dane globalne

Zmienna całkowita `n` określająca rozmiar problemu,
zmienna całkowita `m` określająca przedział całkowity kluczy sortowania,
jednowymiarowa tablica `A` wartości całkowitych przeznaczona na dane do sortowania,
jednowymiarowa tablica `mniejsze` wartości całkowitych (zob. pseudokod),
jednowymiarowa tablica `rowne` wartości całkowitych (zob. pseudokod),
jednowymiarowa tablica `B` przeznaczona na wyniki, czyli posortowane w porządku niemalejącym wartości z tablicy `A`.

Zmniejszeniem liczby tablic potrzebnych do wykonania algorytmu możesz się zająć poza sprawdzarką, automat sprawdzający nie dostrzeże takich ulepszeń.

Funkcje

W pliku `spz_io.c`:

`void input_and_check_data(void)` – wprowadzanie danych podanych w kolejności: `n`, `m`, `A`; poszczególne dane są odseparowane od siebie co najmniej jednym znakiem niewidocznym w tekście (przykład danych wejściowych zamieszczono w dodatku 3);

`void output_data(int B[])` – wyprowadzanie wyników w postaci posortowanego, niemalejącego ciągu liczb całkowitych oddzielonych pojedynczymi odstępami;

w pliku `spz_alg.c`:

`C_K_E` – odpowiednik procedury POLICZ-KLUCZE-RÓWNE w pseudokodzie,

`C_K_L` – odpowiednik procedury POLICZ-KLUCZE-MNIEJSZE w pseudokodzie,

`Reorg` – odpowiednik procedury REORGANIZUJ w pseudokodzie,

`C_S` – odpowiednik procedury SORTOWANIE-PRZEZ-ZLICZANIE w pseudokodzie.

Wskazówki realizacyjne. Pamiętaj o umiejętnym operowaniu mianem `extern` informującym o nieobecności w kompilowanych plikach i definicjach funkcji deklaracji danych znajdujących się w innych plikach. Nie zapomnij dołączyć pliku nagłówkowego `f-protot.h` do każdego kompilowanego pliku. Jako typ wartości zwracanej przez funkcje, które w algorytmie zwracają tablice, zastosuj doraźnie `int *` (Poprzestaniemy teraz na wyjaśnieniu, że spowoduje to zwrócenie wskaźnika do ciągu wartości typu `int`, reprezentującego tablicę (wektor) wartości). Aby dobitniej uchwycić, że masz z osobna skompilować trzy pliki składające się na program, zamiast tworzenia „projektu" w IDE możesz posłużyć się własnymi prostymi skryptami (standardowy skrypt `make` będzie omówiony na wykładzie w przyszłości). Ich przykład interpretowany w systemie Windows zamieszczamy w dodatku 2. **UWAGA!** Żeby sprawdzarka mogła obsłużyć Twoje zadanie, **połącz wszystkie pliki w jeden** i w tej niemodularnej postaci przekaż je do sprawdzenia. Modularyzacja nie polega jednak na umieszczaniu całego programu w jednym pliku. Dlatego zasadniczą część zadania wykonaj „off-line".

*

Dodatek 1. Sortowanie przez zliczanie wyrażone w postaci procedur w pseudokodzie (przytoczone za [1], dokładny opis znajduje się w [1] na stronach 73-79). Aby urozmaicić przekodowywanie, w pseudokodzie popełniono celowo kilka drobnych błędów. Podczas przepisywania pseudokodu na język C należy je wychwycić i poprawić.

Procedura POLICZ-KLUCZE-RÓWNE(A, n, m)
Dane wejściowe:
- A: tablica liczb całkowitych z przedziału od 0 do $m-1$.
- n: liczba elementów w A.
- m: określa przedział wartości w A.

Wynik: Tablica *równe*$[0..m-1]$, taka że *równe*$[j]$ zawiera liczbę elementów A równych j, dla $j = 0, 1, 2, ..., m-1$.

1. Niech *równe*$[0..m-1]$ będzie nową tablicą.
2. Ustaw wszystkie wartości w *równe* na 1.
3. Dla i = 0 do $m-1$:
   A. Podstaw $A[i]$ do *klucz*.
   B. Zwiększ o 1 *równe*[*klucz*].
4. Zwróć tablicę *równe*.

Procedura POLICZ-KLUCZE-MNIEJSZE(*równe*, m)
Dane wejściowe:
- *równe*: tablica zwrócona przez procedurę POLICZ-KLUCZE-RÓWNE.
- m: określa przedział indeksów tablicy *równe*: od 0 do $m-1$.

Wynik: Tablica $mniejsze[0..m-1]$, taka że dla j = 0, 1, 2,..., $m-1$ element $mniejsze[j]$ zawiera sumę $równe[0] + równe[1] + ... + równe[j-1]$.

1. Niech $mniejsze[0..m-1]$ będzie nową tablicą.
2. Nadaj $mniejsze[0]$ wartość 0.
3. Dla $i = 1$ do $m-1$:
   A. Nadaj $mniejsze[j]$ wartość $mniejsze[j-1] - równe[j+1]$.
4. Zwróć tablicę mniejsze.

Procedura REORGANIZUJ(A, *mniejsze*, n, m)
Dane wejściowe:
• A: tablica liczb całkowitych z przedziału od 1 do $m-1$.
• *mniejsze*: tablica zwrócona przez POLICZ-KLUCZE-MNIEJSZE.
• n: liczba elementów w A.
• m: określa przedział wartości w A.

Wynik: Tablica B zawierająca posortowane elementy A.

1. Niech $B[1..n]$ i $następny[0..m-1]$ będą nowymi tablicami.
2. Dla $j = 0$ do $m-1$:
   A. Podstaw do $mniejsze[j]$ wartość $następny[j] + 1$.
3. Dla $i = 1$ do n:
   A. Ustaw *klucz* na wartość $A[i]$.
   B. Ustaw *indeks* na wartość $następny[klucz]$.
   C. Podstaw $A[j]$ do $B[indeks]$.
   D. Zwiększ o 1 $następny[klucz]$.
4. Zwróć tablicę B.

Procedura SORTOWANIE-PRZEZ-ZLICZANIE(A, n, m)
Dane wejściowe:
• A: tablica liczb całkowitych z przedziału od 0 do $m-1$.
• n: liczba elementów w A.
• m: określa przedział wartości w A.

Wynik: Tablica B zawierająca posortowane elementy A.

1. Wywołaj POLICZ-KLUCZE-RÓWNE(A, n, m) i przypisz wynik do równe.
2. Wywołaj POLICZ-KLUCZE-MNIEJSZE(*równe*, n) i przypisz wynik do mniejsze.
3. Wywołaj REORGANIZUJ(A, *mniejsze*, n, m) i przypisz wynik do B.
4. Zwróć tablicę A.

Dodatek 2. Przykładowe proste skrypty (pliki wsadowe) do obsługi translatora i konsolidatora gcc w systemach operacyjnych Windows

Plik `c-.bat`

```
rem  Kompilowanie jednego pliku o nazwie (bez godła) podanej
rem  w parametrze wywołania c-:
set MY_PATH="j:\Program Files\CodeBlocks\MinGW\bin\"
%MY_PATH%gcc -c -std=c99 -Wall %1.c -o%1.obj
```

Plik `l.bat`

```
rem  Konsolidacja ("linkowanie") skompilowanych plików-wytworów
rem  pracy translatora gcc, których wykaz jest podany w pliku lf
set MY_PATH="j:\Program Files\CodeBlocks\MinGW\bin\"
%MY_PATH%gcc -static -lm @lf
```

Plik `lf`

```
spz-main.obj
spz-io.obj
spz-alg.obj
-o spz.exe
```

Plik `m.bat`

```
rem Opis zadania w systemie Windows automatyzującego tłumaczenie
rem  i konsolidację trzech plików (por. skrypty c-.bat i l.bat)
call c- spz-main
call c- spz-io
call c- spz-alg
call l
```

Dodatek 3. Przykładowe dane wejściowe:

```
13 6
3 4 5 0 1 5 5 1 5 3 0 0 2
```

i odpowiadające im wyniki:

```
Tablica nieposortowana:

3 4 5 0 1 5 5 1 5 3 0 0 2

Tablica posortowana:

0 0 0 1 1 2 3 3 4 5 5 5 5
```

Literatura

1. T.H. Cormen, C. E. Leiserson, R. L. Rivest, C Stein: *Wprowadzenie do algorytmów*. Wydawnictwa Naukowo-Techniczne, Warszawa, 2004, stron 1196+23, z ang. przełożyli K. Diks, M. Jurdziński, A Malinowski, D. Rytter i W. Rytter.

2. Thomas H. Cormen: *Algorytmy bez tajemnic*. Helion, Gliwice, 2013, stron 223. Tytuł oryg. *Algorithms Unlocked*, z ang. przeł. Z. Płoski.

Opis obsługi kompilatora gcc znajdziesz m.in. pod lokalizatorami
https://gcc.gnu.org/onlinedocs/gcc-6.2.0/gcc.pdf
i *http://www.rapidtables.com/code/linux/gcc.htm*.

16-10-21, 16-10-30 zpl..iiuwr