w listopadzie 2016

Zdzisław Płoski

Program „Wskaźniki”,

czyli dość prymitywna symulacja
przydziału i zwalniania bloków danych w pamięci
z uwzględnieniem jej reorganizacji w wypadku
nadmiernego pofragmentowania

Zadanie 3 na szóstą pracownię C
IIUWr, 2016

W hierarchii abstrakcji symbolicznych notacji stosowanych w językach programowania wskaźniki i powiązane z nimi operacje zajmują miejsce pośrednie między modyfikacjami adresów argumentów rozkazów asemblerowych, a zapisem tablic, list i innych struktur wysokiego poziomu. W języku C osobny rodzaj wskaźników stanowią jeszcze wskaźniki do funkcji, umożliwiające tablicowanie funkcji lub przekazywanie nazw funkcji w parametrach wywołań, lecz nimi nie będziemy się tutaj zajmować.

Zastosowania wskaźników w języku C istotnie różni się od użycia tablic, ponieważ tablice mają z definicji stałe wymiary, wskaźniki umożliwiają natomiast operowanie na strukturach danych, których wielkość, zależnie od potrzeb, rośnie lub maleje dynamicznie, podczas wykonywania programu. Do pisania takich programów w języku C trzeba prócz wskaźników stosować biblioteczne funkcje dynamicznego przydziału i zwalniania pamięci (`malloc`, `free` itp.). W tym zadaniu zakładamy, że nie będziemy jeszcze z takich funkcji korzystać.

Pisząc ten program, nie używaj notacji symbolizującej indeksowane elementy tablic, tzn. nie stosuj zapisów w rodzaju `A[i]`, `B[j][k]`, `C[p][q][r]`, `for i = 0; i < n; i++` itp. W zamian posłuż się wszędzie, gdzie trzeba, notacją i artymetyką wskaźnikową.

Sprawdzarka nie rozróżni, czy Twój kod będzie wyrażony w jednej, czy drugiej notacji, dlatego kodując program, myśl nie o sprawdzarce (ją zadowoli funkcja napisowa $wynik = f(dane)$), tylko o zakresie równoważności obu zapisów w języku C. Z podobnych przyczyn – sprawdzarka czyta dane tylko z jednego wejścia (`stdin`)– pierwszy parametr programu (rozmiar bufora, o czym dalej), który można by umieścić w poleceniu wykonania programu i potem pobrać w programie, używając odpowiedniego wskaźnika, przeczytamy z tego samego wejścia, co pozostałe dane.

*

Oto Twoje zadanie. Zakładamy, że dynamiczna rezewacja pamięci (użycie funkcji `malloc`) jest niedostępna. Wobec tego zadeklaruj w swoim programie tablicę o rozmiarze określonym pierwszą liczbą w strumieniu danych wejściowych. Tablicę tę potraktujesz jako obszar roboczy do eksperymentów ze wskaźnikami. Do celów testowania swojego programu w sprawdzarce załóż, że rozmiar tego obszaru może mieć zaledwie 1, 2, 3, 10, 100 bajtów itp., lecz może też wynosić kilkadziesiąt, 100000 lub nawet milion bajtów. Na użytek własnych testów zacznij od mniejszego obszaru roboczego, na przykład od dziesięciu bajtów.

Zdeklarowany obszar roboczy będziemy dalej nazywać **buforem**. **Wolnym obszarem** w buforze nazwiemy spójny ciąg wyzerowanych bajtów. Każdy bajt różny od zera (tu: `0x00`) lub spójny ciąg jednakowych bajtów różnych od zera nazywamy

blokiem zajętym. Przez spójny ciąg bajtów rozumiemy bajty o kolejnych adresach z przedziału $[n, m]$, gdzie adresy $n \leqslant m$ mieszczą się w przedziale adresów bufora. Wszystkie funkcje manipulujące na buforze ustawiają i (lub) korzystają ze wspólnego znakowego **wskaźnika położenia w buforze**.

Dalej postępuj według poniższych instrukcji.

1. Zdefiniuj funkcję o prototypie

```
void Clear_Buffer(char* b);
```

zerującą bufor `b`, tzn. wypełniającą go bajtami `0x00`.

2. Zdefiniuj funkcję o prototypie

```
char* Assign_Block(long l);
```

która znajduje w buforze wolny obszar o długości `l` podanej w argumencie jej wywołania i zwraca wskaźnik do początku znalezionego wolnego obszaru. Jeśli w buforze nie ma takiego obszaru, funkcja zwraca `NULL`. Zakładamy dla prostoty, nie dbając o efektywność, że poszukiwanie wolnego obszaru zaczyna się od bieżącej wartości wskaźnika położenia w buforza i przebiega do końca bufora, a potem od początku bufora do bajta poprzedzającego miejsce wskazywane początkowo. To znaczy bufor jest przeszukiwany cyklicznie od aktualnego położenia wskaźnika. Skutki uboczne: funkcja kończy działanie programu z sygnałem

```
"\nAssign_Block: wrong input data\n"
```

jeśli pierwszy argument nie jest większy od zera, jest większy od rozmiaru bufora lub jeśli w buforze nie ma dość miejsca, aby dokonać rezerwacji nawet po uwzględnieniu reorganizacji bufora (zob. p. 6).

3. Zdefiniuj funkcję o prototypie

```
char* Find_Block(char c, long l);
```

znajdującą w buforze blok typu `c` długości `l`. Wyjąwszy poszukiwany typ bloku (por. p. 4) i zmieniony nagłówek sygnału błędu (`Find_Block:` w miejsce `Assign_Block:`), funkcja działa analogicznie jak `Assign_Block` (p. 2). W szczególności obie funkcje ustawiają wskaźnik położenia w buforze na początek znalezionego (lub przydzielonego) bloku.

4. Zdefiniuj funkcję o prototypie

```
void Set_Block(char *b, long l, char c);
```

wypełniającą w buforze blok bajtów `b` o dodatniej długości `l` wartością podaną jako jej trzeci argument. Dopuszczamy tylko wartości będące kodami ASCII wielkich liter alfabetu angielskiego. (Ewentualnie występujące na wejściu małe litery utożsamiamyw programie z wielkimi). Kod znaku, którym jest wypełniany blok, nazywamy **typem bloku**. Funkcja `Set_Block` pozostawia wskaźnik położenia w buforze w pozycji pierwszego bajta, któremu nadaje wartość.

Sprawdzaj, czy podczas zapisywania bloku nie dochodzi do przekroczenia granicy bufora (lepiej to zrobić już w funkcji `Find_Block`). W razie błędu, kończ działanie programu sygnałem:

```
"\nSet_Block: wrong input data\n"
```

5. Zdefiniuj funkcję o prototypie

```
void Release_Block(char *b, long l);
```

która zwalnia blok `b` o długości `l` przez wyzerowanie wszystkich jego bajtów. Funkcja `Release_Block` ustawia wskaźnik położenia w buforze na pierwszy bajt zwolnionego bloku.

Sprawdzaj poprawność danych. Jeśli argument funkcji nie jest dodatni lub zakres zerowania przekracza długość bloku, sygnalizuj:

```
"\nRelease_Block: wrong input data\n"
```

i kończ działanie programu. (Kontrolę tę lepiej wykonać już w funkcji `Find_Block`).

6. Zdefiniuj funkcję o prototypie

```
char* Relocate(void);
```

która przemieszcza wszystkie bloki w buforze w ten sposób, aby bloki zajęte wystąpiły tuż po sobie (bez wolnych obszarów między nimi) w dolnej części bufora, wskutek czego w górnej części bufora powstanie spójny obszar wolny (jeżeli jest to możliwe). Funkcja zwraca wskaźnik do początku obszaru wolnego lub `NULL`, jeśli cały bufor jest wypełniony. Funkcja `Relocate` jest uaktywniana wtedy, gdy w buforze nie znaleziono odpowiednio dużego wolnego bloku bezpośrednio, tj. w wyniku cyklicznego przeszukania jego aktualnego stanu; dopiero po jej wykonaniu, czyli po reorganizacji bufora, można sygnalizować brak miejsca w buforze.

7. Zdefiniuj funkcję o prototypie

```
void Process_Request(void);
```

która będzie czytać porcję danych, tj. długość i typ bloku ze standardowego wejścia (zob. p. 8) i odpowiednio zapisywać nowy blok w buforze lub zwalniać stary. W razie zadeklarowania bloku o zerowej dłogości funkcja sygnalizuje

```
"\nProcess_Request: zero length blocks not allowed\n"
```

i kończy działanie programu.

8. Teraz napisz program, który, korzystając z wyżej określonych funkcji, będzie rezerwował i (lub) zwalniał bloki w buforze, interpretując ciąg danych wejściowych. Składnia tych danych wyrażona w notacji EBNF przedstawia się następująco:

⟨*rozmiar bufora*⟩⟨*znak lub znaki niewidoczne w tekście*⟩
{ ⟨*rozmiar bloku*⟩⟨*typ bloku*⟩⟨*znak lub znaki niewidoczne w tekście*⟩ }*
⟨*rozmiar bloku*⟩⟨*typ bloku*⟩ (terminale: zob. przykłady danych w p. 11).

Dodatni rozmiar bloku oznacza przydział pamięci w buforze (wywołanie funkcji `Assign_Block`, a po niej `Set_Block`). Ujemny rozmiar bloku oznacza zwolnienie bloku (wywołanie funkcji `Release_Block` poprzedzone wywołaniem `Find_Block`). Zerowy rozmiar bloku jest niedopuszczalny i powoduje zakończenie programu (zob. p. 7).

Każdy inny błąd, na przykład żądanie zwolnienia nieistniejącego bloku lub przydziału przekraczającego dostępny sumaryczny wolny obszar w buforze jest dla uproszczenia kwitowane ogólnym sygnałem

`"\n`⟨*nazwa funkcji sygnalizującej błąd*⟩`: wrong input data\n"`

i powoduje zakończenie działania programu.

W przypadku gdy typ nowo przydzielanego bloku jest taki sam jak typ bloku do niego przylegającego, następuje automatyczne połączenie obu bloków w jeden o długości równej sumie ich długości (co tu jest do zaprogramowania?).

W przypadku zwalniania bloku krótszego niż napotkany, program zeruje tylko żądaną, początkową część bloku, pozostawiając resztę bloku bez zmian. Przypominamy, że zwalnianie bloku jest poprzedzone odnajdowaniem bloku, które działa tak jak opisano w p. 3 i 2.

9. Program wyprowadza na wyjściu, wywołując funkcję o prototypie

```
void Output_Buffer_State(void);
```

protokół określający **stan bufora** po wykonaniu wszystkich zleconych na wejściu operacji lub sygnał błędu. Składnia protokołu jest następująca:

{ ⟨*rozmiar bloku*⟩⟨*typ bloku*⟩⟨*spacja*⟩ }*⟨*rozmiar bloku*⟩⟨*typ bloku*⟩

przy czym typ obszarów wolnych jest reprezentowany przez znak `*`.

Przydzielając pamięć, program działa dopóty, dopóki odnajduje wolny blok w buforze, z uwzględnieniem ewentualnej relokacji bloków zajętych (wywołanie funkcji `Relocate`, p. 6).

10. Być może przydadzą Ci się jeszcze funkcje

```
int Buffer_Empty(void);
```

i

```
long Total_Free_Space(void);
```

lecz te szczegóły realizacyjne zależą od Twojego wyboru. Dodatkowo możesz dołączyć do programu funkcję

```
void Print_Buffer(void);
```

drukującą bieżący stan bufora w sposób przytoczony w przykładach 1 i 10 poniżej. Pomoże ona w *uruchamianiu* (ang. *debugging*) programu. Nie wywołuj jednak tej funkcji w wersji programu przeznaczonej dla sprawdzarki.

11. Przykłady danych i wyników

Dane 1

```
10
1A 4B 3Q -4B 2C 2E -1A -2Q 4f 1Z -2E -2F 4T -3T -1F -1C
```

Wyniki 1 (z włączonym śledzeniem bufora – nie dla sprawdzarki)

```
----------
A_________
ABBBB_____
ABBBBQQQ__
A____QQQ__
ACC__QQQ__
ACCEEQQQ__
_CCEEQQQ__
_CCEE__Q__
CCEE___Q__
CCEEQ_____
CCEEQ_____
CCEEQFFFF_
CCEEQFFFFZ
CC__QFFFFZ
CC__Q__FFZ
CCQ____FFZ
CCQFFZ____
CCQFFZTTTT
CCQFFZ___T
CCQ_FZ___T
_CQ_FZ___T
1* 1C 1Q 1* 1F 1Z 3* 1T
```

Dane 2

```
5000
1A 2B 4005Q 13F
151A 77C
```

Wyniki 2

```
1A 2B 4005Q 13F 151A 77C 751*
```

Dane 3

```
10000
1A 2B 4005Q 13F 151A -4005Q 77C -1A
```

Wyniki 3

```
1A 2B 77C 3928* 13F 1* 150A 5828*
```

Dane 4

```
10000
19A 5523B 3047B 11T
```

Wyniki 4

```
19A 8570B 11T 1400*
```

Dane 5

```
1000000
2001T 8000S 700U
```

Wyniki 5

```
2001T 8000S 700U 989299*
```

Dane 6

```
2
1A 1B -1B -1A 2A -1A
```

Wyniki 6

```
1* 1A
```

Dane 7

```
7
3A 4B -3B -3B -1A
```

Wyniki 7

```
Find_Block: wrong input data
```

Dane 8

```
1
1A -1A 1B -1B 1A -1A 1C -1D
```

Wyniki 8

```
Find_Block: wrong input data
```

Dane 9

```
1
1A -1A 1B -1B 1A -1A
```

Wyniki 9

```
1*
```

Dane 10

```
79
1A 2B 20Q 3F 5A 2C 1A 2B 4Q 4F 2A -20Q 7C
-1A 3A 5B 6B 2T 3T 4S 7U 1A 4B 3Q -4B 2C 2E
-1A -2Q 4F 1Z 2Z
```

Wyniki 10 (z włączonym śledzeniem bufora – nie dla sprawdzarki)

```
__________________________________________________________________________________
A_________________________________________________________________________________
ABB_______________________________________________________________________________
ABBQQQQQQQQQQQQQQQQQQQ____________________________________________________________
ABBQQQQQQQQQQQQQQQQQQQFFF_________________________________________________________
ABBQQQQQQQQQQQQQQQQQQQFFFAAAAA___________________________________________________
ABBQQQQQQQQQQQQQQQQQQQFFFAAAAACC_________________________________________________
ABBQQQQQQQQQQQQQQQQQQQFFFAAAAACCA________________________________________________
ABBQQQQQQQQQQQQQQQQQQQFFFAAAAACCABB_____________________________________________
ABBQQQQQQQQQQQQQQQQQQQFFFAAAAACCABBQQQQ_________________________________________
ABBQQQQQQQQQQQQQQQQQQQFFFAAAAACCABBQQQQFFFF_____________________________________
ABBQQQQQQQQQQQQQQQQQQQFFFAAAAACCABBQQQQFFFFAA___________________________________
ABB___________________FFFAAAAACCABBQQQQFFFFAA___________________________________
ABBCCCCCCC_____________FFFAAAAACCABBQQQQFFFFAA___________________________________
ABBCCCCCCC_____________FFF_AAAACCABBQQQQFFFFAA___________________________________
ABBCCCCCCC_____________FFF_AAAACCABBQQQQFFFFAAAAA_________________________________
ABBCCCCCCC_____________FFF_AAAACCABBQQQQFFFFAAAAABBBBB_____________________________
ABBCCCCCCC_____________FFF_AAAACCABBQQQQFFFFAAAAABBBBBBBBBB_________________________
ABBCCCCCCC_____________FFF_AAAACCABBQQQQFFFFAAAAABBBBBBBBBBTT_______________________
ABBCCCCCCC_____________FFF_AAAACCABBQQQQFFFFAAAAABBBBBBBBBBTTTTT_________________
ABBCCCCCCC_____________FFF_AAAACCABBQQQQFFFFAAAAABBBBBBBBBBTTTTTSSSS_____________
ABBCCCCCCC_____________FFF_AAAACCABBQQQQFFFFAAAAABBBBBBBBBBTTTTTSSSSUUUUUUU___
ABBCCCCCCC_____________FFF_AAAACCABBQQQQFFFFAAAAABBBBBBBBBBTTTTTSSSSUUUUUUUA__
ABBCCCCCCCCBBBB________FFF_AAAACCABBQQQQFFFFAAAAABBBBBBBBBBTTTTTSSSSUUUUUUUA__
ABBCCCCCCCCBBBBQQQ_____FFF_AAAACCABBQQQQFFFFAAAAABBBBBBBBBBTTTTTSSSSUUUUUUUA__
ABBCCCCCCCCBBBBQQQ_____FFF_AAAACCABBQQQQFFFFAAAAA____BBBBBBBTTTTTSSSSUUUUUUUA__
ABBCCCCCCCCBBBBQQQ_____FFF_AAAACCABBQQQQFFFFAAAAACC__BBBBBBBTTTTTSSSSUUUUUUUA__
ABBCCCCCCCCBBBBQQQ_____FFF_AAAACCABBQQQQFFFFAAAAACCEEBBBBBBBTTTTTSSSSUUUUUUUA__
ABBCCCCCCCCBBBBQQQ_____FFF_AAAACCABBQQQQFFFFAAAAACCEEBBBBBBBTTTTTSSSSUUUUUUU___
ABBCCCCCCCCBBBB__Q_____FFF_AAAACCABBQQQQFFFFAAAAACCEEBBBBBBBTTTTTSSSSUUUUUUU___
ABBCCCCCCCCBBBB__QFFFF__FFF_AAAACCABBQQQQFFFFAAAAACCEEBBBBBBBTTTTTSSSSUUUUUUU___
ABBCCCCCCCCBBBB__QFFFFZ_FFF_AAAACCABBQQQQFFFFAAAAACCEEBBBBBBBTTTTTSSSSUUUUUUU___
ABBCCCCCCCCBBBB__QFFFFZ_FFF_AAAACCABBQQQQFFFFAAAAACCEEBBBBBBBTTTTTSSSSUUUUUUUZZ_
1A 2B 7C 4B 2* 1Q 4F 1Z 1* 3F 1* 4A 2C 1A 2B 4Q 4F 5A 2C 2E 7B 5T 4S 7U 2Z 1*
```

Dane 11

```
12345
1A 2B 4005Q 13F 151A 77C 1A 2B 400Q 13F 151A -4005Q 77C
-1A 19A 552B 3047B 11T 200T 800S 700U 1A 4B 3Q -4B 2C 2E
-1A -2Q 4F 1Z 2Z
```

Wyniki 11

```
1A 2B 77C 3928* 13F 1* 150A 77C 1A 2B 400Q 13F 170A 2C 2E 3595B
211T 800S 700U 1* 4B 2* 1Q 4F 3Z 2185*
```

Dane 12

```
3
1A 2b 0c
```

Wyniki 12

```
Process_Request: zero length blocks not allowed
```

*

Kiedy już uporasz się ze wszystkim, pomyśl, co należałoby zmienić w tym programie, aby zarządzał on na tych samych zasadach blokami danych innych typów niż `char`. Możesz również ulepszyć (urealnić) administrowanie blokami przez wprowadzenie jednoznacznych identyfikatorów bloków, wykazu (listy) wolnych obszarów, być może z użyciem po jednym bicie na blok, itp. udogodnień.

16-11-11, 16-11-12, 16-11-13, 16-11-15 zpl..iiuwr