Wojciech Myszka

# Laboratorium 5: Tablice. Wyszukiwanie binarne

2018-03-26 21:10:40 +0200

## 1. Tablice

Do tej pory nie było potrzeby odwoływać się do zmiennych złożonych. Programy były bardzo proste i korzystały z niewielkiej liczby zmiennych prostych. Dziś pierwszy raz potrzebować będziemy tablic.

Tablice w języku C to też zmienne, tylko złożone (to znaczy mogą służyć do przechowywania więcej niż jednej wartości). Wszystkie przechowywane wartości muszą być tego samego typu.

### 1.1. Deklaracja tablicy

Tablice, podobnie jak wszystkie zmienne, muszą być deklarowane przed pierwszym użyciem (również inne ograniczenia, na przykład, co do nazwy są identyczne). Deklaracja wygląda tak:

```
typ nazwa[rozmiar];
```

gdzie **typ** to dowolny ze zdefiniowanych w języku C typów zmiennych (**char**, **int**,...), a rozmiar to długość tablicy (czyli, najczęściej, stała) wartość mówiąca ile elementarnych (prostych) danych określonego typu tablica jest w stanie przechować. Zatem

```
int a[10];
```

to tablica o dziesięciu elementach typu **int** nazywająca się a.

Aby uzyskać dostęp do elementu numer k tablicy a używamy zapisu a[k] (co jest odpowiednikiem matematycznego zapisu $a_k$). Numer elementu k nazywany jest **indeksem** tablicy. Indeks **musi być** stałą, zmienną lub wyrażeniem typu całkowitego.

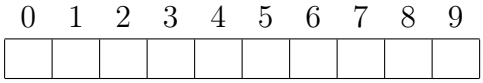
Wpisanie wartości do tablicy:

```
a[3] = 5;
```

„Wybranie” elementu tabicy:

```
k = 3;
u = a[k];
```

Dziesięcioelementowa tablica ma elementy ponumerowane indeksami o wartościach $0, 1, \ldots, 9$. Nie ma możliwości tworzenia tablic o zadanych granicach indeksów (jak, na przykład, w języku Pascal). Schematycznie przedstawia to rysunek 1.




Rysunek 1. Schematyczny obraz tablicy o 10 elementach: dziesięć „pudełeczek” ponumerowanych liczbami od 0 do 9

Język C zupełnie nie przejmuje się czy programista dba o to aby używany indeks tablicy mieścił się w jej granicach. I całą odpowiedzialność za złe działanie programu gdy, ewentualnie, indeksy przekraczają granice tablicy spada na programistę. Przekonać się o tym można uruchamiając poniższy program:

Listing 1. Program generujący błąd przekroczenia dostępnej pamięci

```
#include <stdio.h>
int main(int argc, char **argv)
{
    int a[10];
    int i = 0;
    while ( ++i > 0 )
    {
        a[i] = 0;
        printf("a[%d] = %d\n", i, a[i]);
    }
    return 0;
}
```

Błąd programu polega na niewłaściwym warunku **while** ( ++i > 0 ): nie ogranicza on w żaden sposób zakresu indeksów. Komunikat, który pojawia się na zakończenie działania programu:

```
Segmentation fault
```

oznacza, że program przekroczył przyznany obszar (segment) pamięci. I to system operacyjny dba, żeby działania programu nie miały wpływ na inne programy. A próba zerowania pamięci **poza** obszarem działania programu mogłaby mieć wpływ na inne programy znajdujące się w pamięci operacyjnej komputera. Stąd interwencja systemu operacyjnego, który w trybie awaryjnym zatrzymuje program.

### 1.2. Nadawanie wartości tablicy (inicjalizacja)

Deklarując tablicę nie można mieć pewności, jakie wartości zostaną nadane poszczególnym jej komórkom. Stąd, podobnie jak „zwykłe" zmienne, tablice mogą być inicjalizowane podczas deklarowania:

```
int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

Powyższa instrukcja deklaruje tablicę o nazwie a i w kolejnych jej komórkach wpisuje wartości: 1 do a[0], 2 do a[1], itd.

Wartosci na liście może być mniej niż elementów tablicy

```
int a[10] = {1, 2, 3,};
```

wówczas kolejne komórki przyjmą wartość zero. Na liście nie można podać więcej wartości niż jest elementów tablicy.

Dopuszczalna jest jeszcze jedna forma deklaracji tablicy, bez podawania explicite jej długości:

```
int a[] = {1, 2, 3, 4,};
```

W ty przypadku zadeklarowana zostanie tablica o czterech elementach i zostanie wypełniona kolejnymi wartościami startując od jedynki.

Polecenie **sizeof**() potrafi podać ilość pamięci jaką zajmuje obiekt. Więc **sizeof**( a ) poda liczbę bajtów, które zajmuje tablica a. Każdy element tablicy a jest typu **int** i zajmuje **sizeof**( **int** ) bajtów. Pozwala to wyliczyć liczbę elementów tablicy:

```
int a[] = {1, 2, 3, 4,};
int N;
N = sizeof( a ) / sizeof( int );
```

### 1.3. Tablica jako parametr funkcji

Tablica może być parametrem funkcji. Tablica (raczej) nie może być **wynikiem** funkcji. Przekazując tablicę do funkcji pamiętac należy również o przekazaniu jako kolejnego parametru jej długości.

Poniżej deklaracje dwu funkcji srednia1, srednia2 zawierające dwa możliwe warianty.

```
double srednia1(int N, double t[N]);

double srednia2(int N, double t[ ]);
```

Nie różnią się one niczym (z formalnego punktu widzenia). Użycie tych funkcji jest następujące:

```
. . .
int M = 10;
double dane[M];
double s;
. . .
s = srednia1(M, dane);
. . .
s = srednia2(M, dane);
```

Do programisty należy zadbanie o to, żeby parametry aktualne (prametry wywołania funkcji) były odpowiedniego typu.

## 2. Wyszukiwanie binarne

— Mamy tablicę zawierającą ułożone w kolejności **malejącej**[1] liczby (przyjmijmy, dla skupienia uwagi — całkowite).
— Należy stworzyć algorytm, stosując metodę wyszukiwania binarnego[2], sprawdzający czy zadana liczba $X$ znajduje się w tablicy. Jeżeli tak — algorytm zwraca numer pozycji, na której znajduje się liczba.
— Ogólny schemat blokowy algorytmu znaleźć można wśród slajdów wykładu Technologie Informacyjne „Złożoność obliczeniowa. Trudne zadania." (slajd 33 i następne).

Działanie algorytmu będzie mniej-więcej takie ($D$ to nazwa tablicy z danymi):

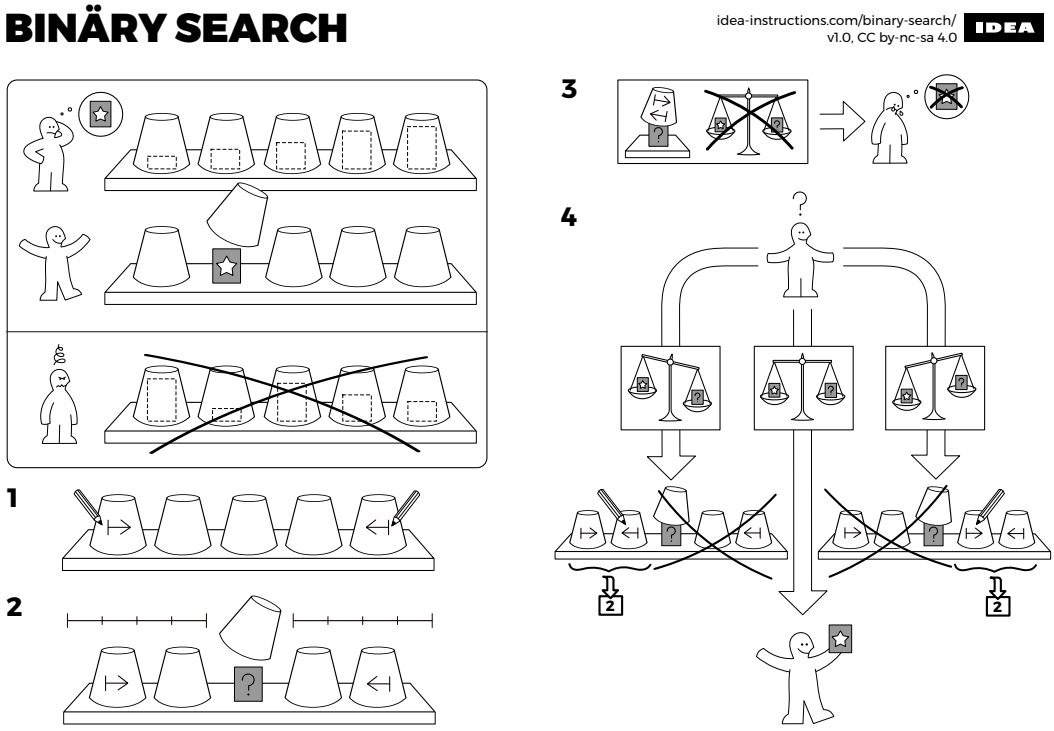
1. Pierwszy element ma indeks 0, zatem $A \leftarrow 0$.
2. Jeżeli tablica ma $N$ elementów to ostatni element ma numer $N - 1$, czyli $B \leftarrow N - 1$.
3. „Środkowy"[3] element tablicy to $C = \frac{A+B}{2}$.
4. Jeżeli $X < D[C]$ wówczas dalsze poszukiwania prowadzić trzeba w pierwszej (dolnej) połówce tablicy, gdy $X > D[C]$ — to w górnej. Gdy uda nam się trafić — to mieliśmy szczęście.

[1] Metoda działa tylko wtedy, gdy dane są ułożone w kolejności: rosnącej lub malejącej.
[2] Analogiczną do metody połowienia z poprzednich zajęć.
[3] Cudzysłów jest niezbędny, gdyż dzielenie odbywa się w sposób całkowitoliczbowy, więc nie zawsze będzie dzieleniem „na pół"!

5. Teraz trzeba tylko odpowiednio zmodyfikować $A$ i $B$ (podobnie jak w poprzednim laboratorium).

Całą trudność algorytmu polega na tym, że w dziedzinie liczb całkowitych trudno w ten sposób przejrzeć wszystkie wartości tablicy. „Przedział" $[A, C]$ nie chce robić się krótszy i krótszy. Co gorsza, sporo algorytmów dostępnych w internecie nie uwzględnia tego faktu...

Na rysunku 2 przedstawiono uproszczoną instrukcję postępowania.




Rysunek 2. Skrócona instrukcja postępowania podczas wyszukiwania binarnego

## 3. Zadania do wykonania

Po zaprogramowaniu algorytmu należy go bardzo dokładnie sprawdzić wybierając liczby z zakresu danych w tabeli (znajdujące się w tabeli i nie występujące w tabeli), spoza tego zakresu. Algorytm powinien działać dla tabeli o długości jeden element, dwa elementy, więcej...

## 4. Wersja PDF tego dokumentu...

...pod adresem.

Wersja: 55 z drobnymi modyfikacjami! data ostatniej modyfikacji 2018-03-26 21:10:40 +0200