Wojciech Myszka

# Laboratorium 12: Anagramy

2016-05-07 09:06:19 +0200

## 1. Zadanie

Program czyta dwa napisy, sprawdzający czy są anagramami (to znaczy występują w nich dokładnie te same litery) i wyświetla o tym informację.

## 2. Algorytm

Jak zwykle można problem rozwiązać na wiele sposobów.

1. Policzyć wystąpienia wszystkich liter w jednym napisie i porównać z wystąpieniami w drugim . (Jeżeli w pierwszym napisie litera a wystąpi dwa razy i litera b trzy razy, a w drugim a trzy razy a b dwa razy — napisy nie mogą być anagramami).
2. Przeglądać kolejne litery w napisie pierwszym i „odznaczać" w drugim. Jeżeli jakaś litera jest w napisie pierwszym, a nie ma w drugim — wyrazy nie są anagramami. Jeżeli po przejrzeniu całego pierwszego napisu zostaną jakieś nieodznaczone litery w drugim — napisy nie są anagramami.
3. Napisy o różnej długości nie mogą być anagramami.
4. ...

## 3. Długość napisu

Funkcja strlen podaje długość napisu:

```
#include <string.h>
...
size_t strlen(const char *s);
```

(Uwaga, typ **size_t** to (najprawdopodobniej) **unsigned long int**.)

Użycie:

```
printf ( "Dlugosc napisu \"Ala ma kota\", to %d\n", \
         (int) strlen("Ala ma kota") );
char * tekst[100];
scanf("%s", tekst);
printf( "Dlugosc wczytanego tekstu to: %d\n", \
        (int) strlen(tekst) );
```

(Ten (**int**) przed wywołaniem funkcji strlen jest po to, aby móc wydrukować wartość **size_t** używając specyfikacji %d.)

## 4. Wielkie/małe litery

Funkcje tolower i toupper dokonują konwersji z wielkich do małych (i odwrotnie). Jeżeli litera jest już „mała” („wielka”) nie ulega zmianie.

```
#include <ctype.h>
...
int toupper(int c);
int tolower(int c);
...
char a = tolower('A');
```

Konwersja dokonywana jest na pojedynczej literze!

Funkcje islower i isupper odpowiadają na pytanie czy litera jest mała/wielka.

```
#include <ctype.h>
...
int islower(int c);
int isupper(int c);
```

Funkcje testują pojedynczy znak!

## 5. Założenia

1. Po pierwsze należy rozstrzygnąć czy program będzie *case sensitive* (czyli czy będzie rozróżniał wielkie i małe litery).
2. Po drugie należy podjąć decyzję, czy program będzie uwzględniał odstępy czy nie. Można rozważyć funkcję, której jednym z parametrów będzie informacja, czy ma uwzględniać odstępy.
3. Zakładamy, że oba napisy będą dostarczane przez użytkownika.
4. Funkcja scanf() nie pozwala (łatwo) wczytać napisu z odstępami.

5. Rozstrzygnąć trzeba czy program wprowadza ograniczenia na długość sprawdzanego tekstu.
6. Ani funkcja scanf() ani gets() nie zwracają uwagi na długość dostarczonego z klawiatury tekstu; nie sprawdzają czy nie przekracza on długości tablicy, do której tekst ma być wczytany. Funkcja scanf() standardowo czyta do pierwszego odstępu co (ale tylko nieco) zmniejsza prawdopodobieństwo przepełnienia bufora.
7. Funkcja fgets() pozwala ograniczyć liczbę czytanych znaków.

## 6. Uzycie funkcji fgets

Prototyp funkcji fgets() wygląda następująco:

```
char *fgets(char *s, int size, FILE *stream);
```

— pierwszy argument funkcji to adres bufora (tablicy znakowj) do którego wpisany zostanie przeczytany z tablicy tekst,
— drugi argument to długość bufora (w znakach),
— trzeci argument to opis (adres struktury danych opisujących) strumienia wejściowego, z którego czytamy; w przypadku standardowego strumienia wejściowego — stdin.

Funkcja fgets czyta tekst ze strumienia. Znak nowej linii zastępowany jest znakiem o kodzie ASCII 0 (koniec tekstu); jeżeli znaków jest więcej niż „długość bufora" czytanych jest tylko tyle znaków, żeby nie przepełnić bufora, a na końcu dodawany jest znak o kodzie ASCII 0.

## 7. Algorytm odczytu napisu

Przedstawiam alternatywny sposób czytania tekstów z terminala. Może on również być wykorzystany (po drobnych modyfikacjach) do czytania danych innego typu o nieznanej długości.

1. Najpierw przydzielana jest początkowa pamięć do bufora danych (linia 20).
2. Algorytm czyta kolejne znaki ze standardowego wejścia używając funkcji getchar() (linia 38). Funkcja ta zwraca kod ASCII przeczytanego znaku lub specjalny kod EOF gdy wystąpi błąd lub system powie, że strumień wejściowy jest już zamknięty[1].
3. Każdy odczytany znak wstawiany jest do bufora (linia 38).

[1] Inaczej mówiąc wystąpi sytuacja podjęcia próby czytania poza końcem pliku.

4. Gdy bufor się wypełni (linia 46) — jego wielkość powiększana jest o kolejny kwant (linia 48).
5. Gdy odczytany znak ma kod '\n' (znak przejścia do nowej linii) lub EOF (koniec danych) (linia 39) w buforze zastępowany jest znakiem o kodzie ASCII 0 (koniec tekstu), a nadmiarowa pamięć jest zwalniana (linia 42).

```
1  /*
2   * napis.c
3   * Copyright 2016 wojciech myszka <myszka@norka.eu.org>
4   */
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8  char * czytaj(void)
9  {
10 /*
11  * dN zawiera informacje o ilości dodawanych bajtów do bufora.
12  * Założymy, że będzie to 10.
13  */
14 #define dN 10
15      int N = dN; // początkowy przydział
16      int i;
17 /*
18  * Przydzielamy funkcją malloc pamięć na 10 znaków
19  */
20      char * bufor;
21      bufor = (char *) malloc(N);
22 /*
23  * Gdy funkcja malloc zwróci wartość NULL
24  * oznacza to, że nie udało się przydzielić
25  * pamięci. Nawet jak taka sytuacja jest mało
26  * prawdopodobna, trzeba ją uwzględnić.
27  * W takim przypadku nie będziemy już czytali
28  * żadnych danych, a funkcja zwróci wartość NULL
29  */
30      if ( bufor == NULL )
31          return bufor;
32 /*
33  * Teraz rozpoczniemy odczyt znaków
34  */
35      i = 0; // Liczba przeczytanych znaków
```

```
36      while ( 1 )
37      {
38          bufor[i] = getchar();
39          if ( bufor[i] == EOF || bufor[i] == '\n' )
40          {
41              bufor[i] = 0; // koniec tekstu
42              bufor = realloc(bufor, i + 1); // zwalniamy nadmiarową pamię
43              return bufor;
44          }
45          i++;
46          if ( i >= N ) // Czy skończyła się przydzielona pamięć?
47          {
48              bufor = realloc(bufor, i + dN);
49              /*
50               * Zwiększamy informację o długości przydzielonego
51               * bufora
52               */
53              N += dN;
54              printf("%p %d\n", bufor, N);
55          }
56      }
57  }
58
59  int main(int argc, char **argv)
60  {
61      char * tekst;
62      char bufor[10];
63      tekst = czytaj();
64      if ( tekst != NULL ) // Sprawdzamy czy coś przeczytane
65          printf("przeczytalem: %d znakow\n \"%s\"\n",
66                  (int) strlen(tekst), tekst);
67      free(tekst); // Zwalniamy przydzieloną pamięć
68      tekst = fgets(bufor, 10, stdin);
69      printf("%p , %p\n", tekst, bufor);
70      printf( "|%s| %d \n", bufor, (int) strlen(bufor) );
71      return 0;
72  }
```

Funkcja zwraca adres początku bufora.

## 8. Wersja PDF tego dokumentu...

... pod adresem.

Wersja: 50 z drobnymi modyfikacjami! data ostatniej modyfikacji 2016-05-07 09:06:19 +0200