П2 – октобар 2016.

аутор: Константин Ђорђевић
vomindoraan@gmail.com

Питалица 1

1) Шта ispisuje sledeći program na programskom jeziku C ukoliko jednostruko ulančana lista sadrži redom brojeve 2 3 1 5 7 4 9? Smatrati da funkcija **`loadList`** ispravno formira, a funkcija **`printList`** ispravno ispisuje sadržaj liste redom od početka.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node {
  int data;
  struct node* next;
} Elem;
Elem* loadList(void);
void printList(Elem*);
void processList(Elem**, int);
void main () {
  Elem *lst = loadList();
  processList(&lst, 4);
  printList(lst);
}
```

```
void processList(Elem **head, int k)
{
  if (k == 0) return;
  Elem* curr = *head; int count = 1;
  while (count < k && curr != NULL) {
    curr = curr->next;
    count++;
  }
  if (curr == NULL) return;
  Elem *node = curr;
  while (curr->next != NULL) curr=curr->next;
  curr->next = *head; *head = node->next;
  node->next = NULL;
}
```

(A) 7 4 9 2 3 1 5 B) 1 3 2 9 4 7 5 C) 1 5 7 4 9 2 3

Уланчана листа је задата на уобичајен начин: поље за податак и показивач на следећег. У функцији `main()` се позива `processList()` и прослеђује јој се пок. на главу листе (пок. на пок. на први члан). Двоструки пок. је зато што се глава листе може променити, па у том случају треба ажурирати спољни пок. `lst`. Двоструки пок. се добија узимањем адресе од `lst` оператором `&`. Поред тога, прослеђује се и број 4.

У функцији `processList()` је `k` = 4, па се не извршава `if`. Помоћни пок. `curr` се поставља на главу листе (`*head` зато што је `head` двоструки пок.), а `count` служи да се броје чланови док се не дође до `k`-тог. `while` петља управо то и ради – помера тренутни унапред док `count` не постане 4 (укупно три померања), а `curr` тада показује на број 5 у листи (па се `if` прескаче). У пок. `node` је запамћена позиција броја 5, а `curr` се затим помера док не дође до последњег члана листе.

Дакле, имамо следеће: `node` показује на број 5 у листи, а `curr` показује на последњи, односно 9. Оно што следи је да се показивачи превежу тако да следећи од 9 буде први члан листе, односно 2, а глава се пребаци да показује на следећи од 5, што је 7. На крају се следећи од `node` (што је сада логички крај листе) постави на `NULL` како листа не би остала везана кружно. Тако се добија: **7 4 9 2 3 1 5**, па је тачно решење под **A**).

Питалица 2

2) Шта treba da stoji umesto `####` i `&&&&` da bi sledeća funkcija na programskom jeziku C vraćala početnu poziciju poslednjeg pojavljivanja stringa **`pat`** u stringu **`txt`**? Funkcija vraća **`-1`** ukoliko se string **`pat`** ne sadrži u stringu **`txt`**.

```
int s(char *pat, char *txt) {
  int i, j, m, n, r = -1;
  m = strlen(pat); n = strlen(txt);
  for (i = 0; i <= n - m; i++) {
    for (j = 0; j < m; j++)
      ####
    &&&&
  }
  return r;
}
```

(A)
```
####:
if (txt[i+j] != pat[j])
  break;
&&&&:
if (j == m) r = i;
```

B)
```
####:
if (txt[i+j] == pat[j])
  continue;
&&&&:
if (i == m) r = j; continue;
```

C)
```
####:
if (txt[i+j] != pat[j])
  break;
&&&&:
if (i == m) r = j; break;
```

Спољашња `for` петља пролази кроз стринг `txt` и означава место одакле креће поређење. Унутрашња петља служи да се `j`-ти (0, 1, 2…) знак стринга `pat` пореди с одговарајућим `j`-тим знаком стринга `txt`, почев од места `i` које је одредила прва петља. На тај начин, ако имамо нпр. `txt` = `"abcdef"` и `pat` = `"cde"`, поређење ће успети када је `i` = 2 а `j` прође по вредностима 0, 1, 2 и одреди да се цео `pat` садржи у `txt`.

Овде одговара кôд под **A)** зато што знамо да се други стринг садржи у првом уколико је `j` прошло од 0 до `m` без прекида и ту се зауставило (исечак `&&&&`). Тада резултат `r` постављамо на индекс одакле смо почели поређење, `i`, што означава да је нађен подстринг који одговара `pat`. Оно што обезбеђује да резултат функције буде последњи такав подстринг је то да се спољашња петља не зауставља када нађе прво појављивање, већ наставља да се врти док не прође кроз цео `txt`.

Одговори под B) и C) немају смисла јер се `m` (дужина стринга `pat`) пореди са `i` уместо са `j`.

Питалица 3

3) Šta će sledeća funkcija, napisana na programskom jeziku C, uraditi sa sadržajem niza **a** dužine **n**?

```
void f(int a[], int n){
   int x=a[n-1];
   if (n>1){
      a[n-1] = a[0];
      f(a+1, n-2);
      a[0] = x;
   }
}
```

A) Iz niza će izbaciti svaki drugi element.
B) Elementi niza će biti obrnuti dva puta (ostaće u početnom poretku).
(C) Redosled elemenata niza će biti obrnut.

Ово је рекурзивна функција, што се може видети из пете линије. `x` је помоћна променљива у којој се чува вредност тренутног последњег члана низа. На четвртој линији видимо да се на последње место преписује први члан (`a[n-1] = a[0]`), а на шестој да се на прво место преписује последњи (`a[0] = x`). Дакле, први и последњи мењају места – обрћу се.

Пошто се у наредном позиву прослеђује низ почев од следећег, другог елемента (ако је `int a[]` исто што и показивач на први члан, онда је `a+1` показивач на други), обрнуће се други и претпоследњи, онда трећи и `n-3`. члан и све тако, идући ка средини. Кључно је да се прослеђује `n-2` а не `n-1` јер би то значило да се лева граница помера а десна увек остаје иста (последњи члан низа). Дакле, тачан одговор је под **C)**.

Питалица 4

4) Šta ispisuje sledeći program na programskom jeziku C? Program se nalazi u datoteci **obrada.exe** i pokreće se sledećom komandom: **obrada bundeva dubrava dabar 1 2 3**

```
#include <stdio.h>
#include <stdlib.h>
void main(int n, char* a[]) {
   int i, x = atoi(a[n-1]);
   char *t = a[x];
   x = atoi(a[n-1-x]);
   for(i = 0; a[x][i]; i++) {
      int j = 0;
      for(j = 0; *(t) != 0; j++) {
         if (a[x][i] == *t++)
         printf("%c",*t?*t:'_');
      }
      t -= j;
   }
}
```

A) rav_uv_ **(B)** a_brabr C) aa_

Програм се позива с аргументима командне линије. Уместо уобичајеног `int argc, char *argv[]`, овде се променљиве које садрже број аргумената и њихове текстуалне представе зову `n` и `a`, респективно, али то не мења ништа. Први аргумент, `a[0]`, је увек име програма (у овом случају `"obrada"`), а остатак низа садржи редом остале аргументе као стрингове (`1` је стринг, а не `int`).

Овде је `n` = 7, а у `x` се памти вредност последњег аргумента, 3, претворена у број помоћу `atoi()`. Стринг `t` се ставља да показује на `a[3]`, што је четврти члан низа, односно `"dabar"`. На крају левог дела се у `x` уписује бројна вредност члана `a[7-1-3]`, што је `"dabar"`, али пошто то није број уписаће се 0.

Приметимо да се у наставку `x` нигде не мења. То значи да ће услов прве `for` петље увек бити `a[0][i]`, тј. испитиваће се да ли је `i`-ти знак стринга `a[0]` (`"obrada"`) различит од 0. Унутрашња петља пролази

по стрингу `t` (`"dabar"`) док не дође до његовог краја и сваки пут `t` помера на наредни знак (у изразу `*t++` се `++` се односи на пок.). У телу петље се проверава да ли је неки од тих знакова једнак тренутном знаку `a[0][i]`, па ако јесте долази до исписа. Међутим, знак који се испише није тај који је проверен, већ један после њега, зато што се пок. инкрементира у току провере и већ на следећој линији има нову вредност (да је писало `*++t`, исписивао би се исти знак који се проверава). У случају да се дошло до краја стринга, тј. до нултог знака, уместо њега се штампа `'_'` (`*t?` пита да ли је знак на који показује `t` различит од 0).

На крају се `t` враћа за `j` места уназад, односно на почетак стринга `"dabar"`, чиме се припрема за наредну итерацију окружујуће петље.

Сад можемо проћи кроз програм и видети шта се исписује. Прво се знаци пореде са `'o'`, па пошто нема поклапања, ништа се не штампа. У другом пролазу се знаци пореде са `'b'`, па пошто ту има поклапање, исписаће се знак иза `'b'` у `"dabar"`, што је `'a'`. Потом се проверава `'r'` и ту је услов опет испуњен, али како је то последњи знак у стрингу и иза њега се налази `'\0'`, исписаће се `'_'`. Овако се наставља до краја и добија се **a_brabr**, што је одговор под **B)**.