Prelucrarea şirurilor de caractere. Funcţii. Aplicaţii.

Resposabili:Responsabili:

- Dorinel Filip (CA 2016-2020) [mailto:ion_dorinel.filip@cti.pub.ro]
- Darius Neațu (CA 2019-2020) [mailto:neatudarius@gmail.com]
- Emil Racec (2012) [mailto:emil.racec@gmail.com]
- Alina Simion (2008) [mailto:alina.g.simion@gmail.com]
- Ştefan Bucur (2006) [mailto:stefan.bucur@gmail.com]

Obiective

În urma parcurgerii acestui laborator studentul va fi capabil:

- să declare şi să folosească şiruri de caractere
- să folosească funcţiile de manipulare a şirurilor de caractere din libraria string.h

Noţiuni teoretice

Şiruri de caractere

Un **caracter** se declară în C de forma: `char a='a';` Pentru inițializarea lui, se observă că am pus un caracter între apostroafe.

Un **şir de caractere** presupune practic un vector de caractere, terminat prin caracterul `\0`.

Compilatorul folosește în mod implicit această reprezentare, astfel încât cea mai simplă declarație este `char c[]="cuvant";` Observăm aici folosirea ghilimelelor în locul apostroafelor. Acecastă instrucțiune va aloca un spațiu de 7 octeți pe care va reprezenta șirul de caractere 'cuvant' (care are 6 caractere).

Dacă dorim să alocăm un spațiu de memorie mai mare (pentru a putea folosi variabila pentru a stoca șiruri de caractere mai lungi), putem folosi o declarație de tipul `char c[10] = "cuvant";`. Astfel am alocat spațiu suficient pentru un șir de 9 caractere.

Deși o inițializare de tipul `char c[6] = "cuvant"`, în care spațiul alocat este egal cu numărul de caractere, nu va determina compilatorul să genereze un warning/o eroare, acest lucru poate avea rezultate neașteptate dacă în memorie - la finalul șirului - nu se află (întâmplător) valoarea 0 binar.

Cum s-a prezentat anterior, o variabilă vector conţine adresa de început a vectorului(adresa primei componente a vectorului), şi de aceea este echivalentă cu un pointer la tipul elementelor din vector. Deci declaraţiile de mai jos vor declara fiecare cate un şir de caractere:

```
char a[5];
char *b="unsir";
char *c;
```

Diferenţa majoră dintre între ele este însa că primele două declaraţii vor aloca 5 pozitii în memorie, pe când ultima nu va aloca nici o zona de memorie, necesitând sa fie ulterior alocată, folosind funcţiile de alocare dinamică (`malloc(), calloc(), realloc()`), prezentate în laboratorul anterior.

Între prima și a 2-a declarație, diferența este mai subtilă și constă în faptul că declarația `char *b="unsir"` va determina compilatorul să plaseze șirul respectiv într-o zonă de memorie asupra căreia nu avem drepturi de scriere, deci orice încercare de a modifica acel șir - pe Linux - va genera, cel mai probabil, o eroare de tipul `segmentation fault`.

Un mic exemplu de citire a unui şir, caracter cu caracter pana la întâlnirea caracterului `-`:

```
#include <stdio.h>
#include <string.h>

#define N 30

int main () {
   char str[N], c;
   int n = 0;

   do {
     scanf("%c", &c);
     if (c == '-') {
       break;
     }
     str[n++] = c;
   } while(1);

   str[n] = '\0'; // setam terminatorul de şir
   printf("%s", str);

   return 0;
}
```

Pentru citirea si afisarea unui şir de caractere se poate folosi flagul '**s**' la citirea cu `scanf` sau afişarea cu `printf`. De asemenea biblioteca `stdio.h` defineşte funcţiile `gets()` şi `puts()` pentru lucrul cu şiruri de caractere.

- `gets(zona)` -- citeşte de la terminalul standard un şir de caractere terminat cu linie noua (enter).
  - Funcţia are ca parametru adresa zonei de memorie în care se introduc caracterele citite.
  - Funcţia returneaza adresa de început a zonei de memorie.
- `puts(zona)` – afişeaza la terminalul standard şirul de caractere din zona data ca parametru, până la caracterul terminator de şir (\0), în locul căruia va afișa caracterul sfârșit de linie.
  - Funcţia are ca parametru adresa zonei de memorie de unde începe afişarea caracterelor.
  - Funcţia returneaza codul ultimului caracter (diferit de \0) din şirul de caractere afişat şi -1 dacă a aparut o eroare.

Funcţia `gets()` va citi de la tastatura câte caractere sunt introduse, chiar daca şirul declarat are o lungime mai mică. Presupunem un şir declarat: `char a[]="unsir"` , care va avea deci 5 caractere. Citind un şir de lungime mai mare ca 5 de la tastatura, în şirul a, la afişare vom vedea ca s-a reţinut tot sirul (nu

doar primele 5 caractere)! . Nimic deosebit până acum. Dar dacă luăm în considerare că citirea caracterelor auxiliare se face în continuare în zona de memorie, ne punem problema ce se va suprascrie?! Raspunsul este: nu se ştie… poate nimic important pentru programul nostru, poate ceva ce il va bloca sau duce la obţinerea de date eronate.

Pentru a evita aceasta se recomandă utilizarea `fgets()`.

- `fgets(zona, lung_zona, stdin)` -– citeşte de la `stdin` un şir de caractere terminat printr-o linie nouă dacă lungimea lui este mai mică decat `lung_zona` sau primele `lung_zona - 1` caractere în caz contrar. Parametrii sunt: zona de memorie, lungimea maxima admisă a şirului, şi terminalul standard de intrare. În cazul în care şirul dorit are lungime mai mică decât cea maximă, înaintea terminatorului de şir (\0), în zona de memorie va fi reţinut şi enter-ul dat(\n).

Funcţii din <string.h>

Pentru manipularea şirurilor de caractere în limbajul C se folosesc funcţii declarate în fişierul <string.h>. Vom încerca să le detaliem putin pe cele mai des folosite:

strlen()

```
size_t strlen(const char *str);
```

Returneaza lungimea unui şir dat ca parametru. (numarul de caractere până la întalnirea terminatorului de şir: \0)

Exemplu:

```
#include <stdio.h>
#include <string.h>

#define N 256

int main () {
  char text[N];
  printf("Introduceti un text: ");
  gets(text);
  printf("Textul are %u caractere.\n", strlen(text));
  return 0;
}
```

Iesire:

```
Introduceti un text: just testing
Textul are 12 caractere.
```

memset()

```
void* memset(void *ptr, int val, size_t num);
```

În zona de memorie dată de pointerul ptr, sunt setate primii num octeți la valoarea dată de val. Pentru șiruri de caractere - în care fiecare element ocupă 1 octet - aceasta are ca rezultat înlocuirea primelor 'num' valori cu cea dată ca argument.

Funcţia returnează şirul ptr.

Exemplu:

```
#include <stdio.h>
#include <string.h>

int main () {
  char str[] = "nu prea vreau vacanta!";
  memset(str, '-', 7);
  puts(str);
  return 0;
}
```

Iesire:

```
------- vreau vacanta!
```

Ca programatori, veți întâlni adesea situații în care `memset` este folosit pentru inițializarea de vectori de diverse tipuri.

Atunci când scrieți sau evaluați cod care face acest lucru trebuie să aveți în vedere că memset face scrierea valorii primite pe fiecare **octet** (fără a ține cont de dimensiunea reperezentării tipului de date). Următorul cod arată cum putem folosi memset pentru a inițializa un vector de int la 0 folosind memset, respectiv care ar fi rezultatul dacă am încerca să inițializăm vectorul cu o altă valoare:

```
#include <stdio.h>
#include <string.h>

#define SIZE 2

int main(void)
{
    int a[SIZE], b[SIZE], i;
    memset(a, 0, SIZE * sizeof(int));
    memset(b, 5, SIZE * sizeof(int));

    for(i = 0; i < SIZE; i++)
        printf("%d ", a[i]);

    printf("\n");

    for(i = 0; i < SIZE; i++)
        printf("%d ", b[i]);

    return 0;
}
```

Rezultatul este…

```
0 0
84215045 84215045
```

rezultatul neașteptat de pe a 2-a linie provenind din faptul că fiecare octet al int-urilor a fost setat la 5, și nu valoarea întregii structuri.

Este de menționat faptul că utilizarea `memset` în astfel de situații **nu** este recomandată.

memmove()

```
void* memmove(void *destination, const void *source, size_t num);
```

Copiază un număr de num caractere de la sursă, la zona de memorie indicată de destinaţie. Copierea are loc ca şi cum ar exista un buffer intermediar, deci sursa si destinatia se pot suprapune. Funcţia nu verifică terminatorul de şir la sursă, copiază mereu num bytes, deci pentru a evita depăsirea trebuie ca dimensiunea sursei sa fie mai mare ca num. Funcţia returnează destinaţia.

Exemplu:

```
#include <stdio.h>
#include <string.h>

int main () {
  char str[] = "memmove can be very useful.....";
  memmove(str + 20, str + 15, 11);
  puts(str);
  return 0;
}
```

Iesire:

```
memmove can be very very useful.
```

memcpy()

```
void* memcpy(void *destination, const void *source, size_t num);
```

Copiază un număr de num caractere din şirul sursă in şirul destinaţie. Funcţia returnează şirul destinaţie.

Exemplu:

```
#include <stdio.h>
#include <string.h>

#define N 40

int main () {
  char str1[] = "Exemplu";
  char str2[N];
  char str3[N];
  memcpy(str2, str1, strlen(str1) + 1); // + 1 este necesar pentru a copia și terminatorul de șir
  memcpy(str3, "un sir", 7);
  printf("str1: %s\nstr2: %s\nstr3: %s\n", str1, str2, str3);
  return 0;
}
```

Iesire:

```
str1: Exemplu
str2: Exemplu
str3: un sir
```

strcpy()

```
char* strcpy(char *destination, const char *source);
```

Copiază şirul sursă in şirul destinaţie. Şirul destinaţie va fi suprascris. Funcţia asigură plasarea terminatorului de şir în şirul destinaţie după copiere. Funcţia returneaza şirul destinaţie.

strncpy()

```
char* strncpy(char *destination, const char *source, size_t num);
```

Asemeni cu `strcpy()`, dar in loc de a fi copiată toata sursa sunt copiate doar primele *num* caractere.

Exemplu:

```
#include <stdio.h>
#include <string.h>

#define N 40

int main () {
  char str1[] = "Exemplu";
  char str2[N];
  char str3[N];
  strcpy(str2, str1);
  strncpy(str3, "un sir", 2);
  str3[2] = '\0';
  printf("str1: %s\nstr2: %s\nstr3: %s\n", str1, str2, str3);
  return 0;
}
```

Iesire:

```
str1: Exemplu
str2: Exemplu
```

```
str3: un
```

strcat()

```
char* strcat(char *destination, const char *source);
```

Concatenenaza şirul sursă la şirul destinaţie. Funcţia returnează şirul destinaţie.

Șirul destinație trebuie să aibă suficientă memorie alocată pentru a a acomoda șirul rezultat.

strncat()

```
char* strncat(char *destination, const char *source, size_t num);
```

Asemeni cu `strcat()`, dar în loc de a fi concatenată toată sursa sunt concatenate **cel mult** primele *num* caractere din șirul sursa (aceasta putând fii și mai scurt).

Exemplu:

```
#include <stdio.h>
#include <string.h>

#define N 80

int main () {
  char str[N];
  strcpy(str, "ana ");
  strcat(str, "are ");
  strcat(str, "mere ");
  puts(str);
  strncat(str, "si pere si prune", 7);
  puts(str);
  return 0;
}
```

Iesire:

```
ana are mere
ana are mere si pere
```

strcmp()

```
int strcmp(const char *str1, const char *str2);
```

Compară şirul *str1* cu şirul *str2*, verificându-le caracter cu caracter. Valoarea returnată este 0 daca cele şiruri sunt identice, mai mare ca 0 daca str1 este "mai mare"(alfabetic) şi mai mic ca 0 altfel.

Exemplu:

```
#include <stdio.h>
#include <string.h>

#define N 80

int main () {
  char cuv[] = "rosu";
  char cuv_citit[N];
  do {
    printf ("Ghiceste culoarea...");
    gets(cuv_citit);
  } while (strcmp(cuv,cuv_citit) != 0);

  puts("OK");

  return 0;
}
```

În situația în care șirurile au lungimi diferite, ultima comparație se face între \0 și caracterul de pe aceași poziție din șirul mai lung,

strchr()

```
char* strchr(const char *str, int character);
```

Caută caracterul *character* în şirul *str* şi returnează un pointer la *prima* sa apariţie sau NULL dacă acesta nu a fost găsit..

strrchr()

```
char* strrchr(const char *str, int character);
```

Caută caracterul *character* în şirul *str* şi returnează un pointer la *ultima* sa apariţie sau NULL dacă acesta nu există în şir.

strstr()

```
char* strstr(const char *str1, const char *str2);
```

Caută şirul *str2* în şirul *str1* şi *returnează un pointer la prima sa apariţie*, sau *NULL* dacă nu a fost găsit.

strdup()

```
char* strdup(const char *str);
```

Realizează un duplicat al şirului *str*, pe care îl şi returnează. Spațiul de memorie necesar copiei este alocată dinamic, fiind responsabilitatea noastră să o dealocăm (așa cum am s-a prezentat laboratorul anterior).

Exemplu:

```
#include <stdio.h>
#include <string.h>

#define N 80

int main () {
  char str[N] = "salut", *d;

  d = strdup(str);
  if(d == NULL) {
      printf("Eroare!\n");
      return -1;
  }

  puts(d);
  free(d);

  return 0;
}
```

`strdup(..)` va aloca întotdeuna `strlen() + 1` octeți pentru destinație, indiferent de dimensiunea memoriei alocate pentru sursă.

strtok()

```
char* strtok(char *str, const char *delimitators);
```

Funcţia are rolul de a împarţi şirul *str* în tokens(subşiruri separate de orice caracter aflat în lista de delimitatori), prin apelarea ei succesivă.

La primul apel, parametrul *str* trebuie sa fie un şir de caractere, ce urmează a fi împartit. Apelurile urmatoare, vor avea în loc de *str*, `NULL` conţinuând împarţirea aceluiaşi şir.

Funcţia va returna la fiecare apel un token(un subsir), ignorând caracterele cu rol de separator aflate în şirul de delimitatori. O dată terminat şirul, funcţia va returna `NULL`.

Implementarea curentă din `<string.h>` nu permite folosirea `strtok()` în paralel pe mai mult de un şir.

Exemplu:

```
#include <stdio.h>
#include <string.h>

int main () {
  char str[] = "- Uite, asta e un sir.";
  char *p;
  p = strtok(str, " ,.-");
  /* separa sirul in "tokeni" si afiseaza-i pe linii separate. */
  while (p != NULL) {
    printf("%s\n", p);
    p = strtok(NULL, " ,.-");
  }

  return 0;
}
```

Iesire:

```
Uite
asta
e
un
sir
```

Exercitii laborator CB/CD

Primul exercitiu presupune modificarea/adaugarea de instructiuni unui cod existent pentru a realiza anumite lucruri. In momentul actual programul numara cate cuvinte doar cu litere mici se gasesc intr-un sir de caractere.

ex.c

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>

#define SEPARATORS " ,."

int count_lowercases_words(char *str)
{
        char *tmp_str = strdup(str);
        unsigned int count = 0;

        if (tmp_str == NULL) {
                printf("Eroare la alocare\n");
                return -1;
        }

        char *word = strtok(tmp_str, SEPARATORS);
        while (word) {
            unsigned int i;

            for (i = 0; i < strlen(word); i++) {
                    if (isupper(word[i])) {
                            break;
                    }
            }
```

```
            if (i == strlen(word)) {
                count++;
            }

            word = strtok(NULL, SEPARATORS);
        }

        free(tmp_str);
        return count;
}

int main(void)
{
        char sentence[] = "Ana are mere, pere si gutui. Gigel nu are nimic.";

        printf("%d\n", count_lowercases_words(sentence));

        return 0;
}
```

Cerinte:

- Sa se numere cate cuvinte au cel putin un caracter din sirul "api".
- Sa se realizeze o functie care intoarce un sir de caractere compus din cuvintele care incep cu litera mare.

Următoarele două probleme vă vor fi date de asistent în cadrul laboratorului.

Checker laborator 10 [https://drive.google.com/drive/folders/1qB6EZLGVubKbuTXMtMue06egH_8fo25M]

Tutorial folosire checker laborator [https://ocw.cs.pub.ro/courses/programare/checker]

Exercitii de laborator

1. [2p] Să se citească o succesiune de cuvinte. Să se creeze o funcţie: `void ordCresc(char *vectorschar[], int n);` care să ordoneze cuvintele crescător
   - după lungimea acestora.
   - alfabetic
2. [2p] Să se determine dacă o propozitie este palindromă. O propozitie este palindromă daca citită de la prima literă pană la ultima are aceeasi succesiune ca citită de la ultima literă până la prima. Nu conteaza dacă sunt litere mici sau mari.
   Exemplu: ele fac cafele
3. [1p] Folosind funcţia strtok, citiţi un şir de caractere şi afişaţi pe ecran cuvintele sale constituente.
4. [2p] Folosind funcţia strtok, citiţi un şir de caractere, apoi un cuvânt şi afişaţi pe ecran numărul de apariţii al cuvântului în şir.
5. [3p] Folosind funcţiile din `<string.h>` înlocuiţi într-un text dat o secventă de caractere cu altă secvenţă de caractere, date la intrare.

Bonus

1. [3p] Se dă N de la tastatură. Citiţi pentru fiecare N numele, prenumele şi vârsta elevului respectiv, alocând întreaga memorie în mod dinamic pentru fiecare elev nou (lungimea numelui şi a prenumelui se consideră a fi 20 de caractere, iar vârsta se consideră între 0 şi 100). Afişaţi pe ecran, în funcţie de opţiunea selectata, lista citită de la tastatură, ordonată în funcţie de nume, prenume, respectiv vârsta.

Referinţe

- Wikipedia - String(Computer science) [http://en.wikipedia.org/wiki/String_(computer_science)]
- Wikipedia - C string handling [http://en.wikipedia.org/wiki/String.h]

Probleme laborator 14-16 [https://drive.google.com/open?id=1n-X0kssdqDXF9i73VfMxAb7GZP7CzhT3]

programare/laboratoare/lab10.txt · Last modified: 2020/10/05 00:38 by darius.neatu