Programarea Calculatoarelor Cursul 8

Șiruri de caractere

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Obiective

- Cum se reprezintă un text?
  - șiruri de caractere sau string-uri
- Care sunt funcțiile importante pentru manipularea string-urilor?
- Analiză problemă căutare subșir

Șiruri de caractere, string-uri

- Definiție
  - Sunt tablouri de `char` terminate cu un caracter special
- Caracterul nul
  - caracterul cu cod ASCII 0 sau echivalent `'\0'`
  - a nu se confunda cu pointerul `NULL`
  - La afișare se afișează toate caracterele până la primul caracter nul începând de la adresa șirului
- În alte limbaje se reprezintă prin șir + lungime
  - avantaj - nu e nevoie de parcurgere pentru a ști lungimea
  - dezavantaj - ocupă puțin mai mult spațiu

String-uri reprezentare în memorie

- caracterele string-ului împreună cu caracterul nul se stochează într-o zonă de memorie continuă - ca la orice tablou unidimensional
- fiecare caracter se stochează pe un byte
- la alocare dinamică trebuie rezervat loc și pentru caracterul nul

Declarare string-uri

- Varianta 1
  - Declarare ca tablou cu dimensiune dată
  - Se alocă spațiu pe stivă (sau global) pentru `dim` caractere
  - Permite stocarea unui string de lungime dim-1 fiindcă avem nevoie de o poziție pentru `'\0'`
  - Recomandat ca zona să fie mai extinsă în cazul în care se modifică stringul (se adaugă alte caractere)
  - *s* este de fapt un pointer constant la primul caracter
    - nu se poate schimba unde arată pointerul

```
char s[dim];
```

Declarare string-uri

- Varianta 2
  - Declarare ca tablou cu dimensiune necunoscută
  - Dimensiunea se determină din string-ul cu care se inițializează
    - obligatoriu se inițializează
  - Se alocă spațiu pe stivă (sau global) pentru caracterele necesare și caracterul `'\0'`
  - Nu se poate adăuga caractere după
  - *s* este de fapt un pointer constant la primul caracter
    - nu se poate schimba unde arată pointerul

```
char s[] = "init";
```

Declarare string-uri

- Varianta 3
  - Declarare ca pointer la un caracter
  - Arată la adresa de unde începe șirul
  - Dacă se inițializează
    - String-ul se alocă într-o zonă dedicată string-urilor pentru caracterele necesare și caracterul `'\0'`
  - Nu se poate adăuga caractere după
  - s este pointer la caractere constante
    - nu putem modifica string-ul
    - s[0] = 'x' - ilegal

```
char* s = "init";
```

Inițializare string-uri

- Un string se poate inițializa ca orice tablou de variabile
  - sintaxa cu șir de caractere încadrate între ghilimele simple
  - trebuie să specifice explicit caracterul nul

```
char s[] = {'a', 'b', '\0'};
```

- Există o modalitate mai compactă (prezentată și anterior)
  - sintaxa cu ghilimele duble introduce automat caracterul nul

```
char s[] = "ab";
```

- În cazul în care un string rămâne neinițializat putem să copiem un alt string folosind
  - funcția `strcpy` dacă stringul este tablou
  - atribuire dacă stringul este pointer

```
char s[20];
strcpy(s, "ab");
```

```
char* s;
s = "ab";
```

Lungimea și mărimea

- Lungimea unui string este numărul de caractere
  - se poate obține folosind funcția `strlen`
  - în mod echivalent: este poziția primului caracter nul
  - se folosește indexare de la 0
  - pentru aflarea lungimii este nevoie de o parcurgere
- Mărimea zonei ocupate în memorie
  - se poate obține folosind operatorul `sizeof`
  - numărul de bytes sau octeți ocupați
  - pentru string-uri definite ca tablou
    - include caracterele și caracterul nul
  - pentru string-uri definite ca pointeri
    - se referă la mărimea pointerului
    - constant pentru un sistem de operare + compilator
      - 4 bytes pentru Windows și gcc pe 32 biți

Lungime vs. mărime string - exemplu simplu

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

void f(char s[]){
    printf("sz %d\n", sizeof(s));
}
int main(){
    char s[] = "a";
    f(s);
    printf("sz %d\n", sizeof(s));
    printf("len %d\n", strlen(s));

    char s2[20] = "a";
    printf("sz %d\n", sizeof(s2));
    s2[1] = 'b';
    printf("sz %d\n", sizeof(s2));
    printf("len %d\n", strlen(s2));
    return 0;
}
```

- includem biblioteca string.h

- funcție care primește un string
- afișează dimensiunea unui pointer nu dimensiunea tabloului original

- declarare varianta 2

- afișează 2 - s ocupă 2 bytes - include `'\0'`
- afișează 1 - s are un singur caracter

- declarare varianta 1
- afișează 20 - am stabilit dimensiunea
- suprascriem caracterul nul
- tot 20 - neafectat
- afișează 2 - s2[2] este `'\0'` din cauza inițializării

Conversie din și în string-uri

- În general, pentru a converti un string într-un alt tip se folosește `sscanf` - citire formatată din string
- Există funcții specifice pentru `int`, `float` și `long long` `atoi, atof, atoll`
- Pentru a converti un alt tip într-un string se folosește `sprintf` - scriere formatată într-un string

Parsare string-uri - exemplu simplu

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

int main(){
    float x = 1.7;
    char s[20];
    sprintf(s, "%fxx", x);
    puts(s);

    float y;
    sscanf(s, "%f", &y);
    printf("%f\n", y);

    sscanf(" 1.56/-9", "%f/%f", &x, &y);
    printf("%f %f\n", x, y);
    return 0;
}
```

- includem biblioteca string.h

- din float în string
- 1.7xx

- se oprește la primul caracter care nu se potrivește
- 1.7

- este ușor să parsăm un string fără să-l separăm pe bucăți cu sscanf

Parsare string-uri - exemplu complicat

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

int main() {
  char* s = "15/6 7/128 1/1";
  while (*s) {
    int x, y, n;
    sscanf(s, "%d/%d %n", &x, &y, &n);
    printf("%d/%d %d\n", x, y, n);
    s += n;
  }
  return 0;
}
```

- includem biblioteca string.h
- definim un string cu fracții
- cât timp nu am ajuns la caracter nul
- citim *x, y* și salvăm în *n* câte caractere am citit din *s* până la *%n*
- Mutăm pointerul cu *n* poziții

Funcții din ctype.h

- isdigit - este cifră
- isalpha - este literă
- isupper - este majusculă
- islower - este literă mică
- toupper - transformă în literă mare
- tolower - transformă în literă mică
- isspace - este spațiu - include tab `'\t'`, linie nouă `'\n'`

https://en.cppreference.com/w/cpp/header/cctype

Funcții din string.h

- strlen - lungimea
- strcpy, strncpy - copiază (n caractere)
- strcat, strncat - concatenează
- strdup - alocă dinamic și copiază
- strcmp, strncmp, stricmp, strnicmp - compară
- strchr - caută un caracter într-un string
- strstr - caută un substring într-un string
- strtok - separă în bucăți

https://en.cppreference.com/w/c/string/byte

Funcția strlen

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

- Determină lungimea unui string prin căutarea primului caracter nul începând de la adresa s
- Parametrii
  - `const char* s` - șirul de caractere
- Valoare returnată
  - Lungimea stringului s
- `size_t` este echivalent cu `unsigned long`
- Erori posibile
  - caracterul nul de terminare nu este definit și se accesează zone interzise de memorie

Funcția strcpy

```
char* strcpy(char* dest, const char* src);
```

- Copiază stringul de la sursa `src` la destinația `dest` incluzând caracterul nul
- Parametrii
  - `char* dest` - șirul de caractere de destinație
  - `const char* src` - șirul de caractere sursă
- Valoare returnată
  - O copie a pointerului `dest` în caz de succes
  - În caz de eșec se setează `dest[0]` la 0 dacă `dest` nu este pointerul NULL
- Erori posibile
  - `dest` nu este pointer valid
  - zona alocată de la `dest` nu este suficient de mare
  - cele două zone de memorie se suprapun
  - sursa nu are caracter nul

Funcția strcmp

```
int strcmp(const char* lhs, const char* rhs);
```

- Determină ordinea lexicografică între o pereche de string-uri
- Parametrii
  - `const char* lhs` - șirul de caractere din partea stânga
  - `const char* rhs` - șirul de caractere din partea dreapta
- Valoare returnată
  - număr negativ — în cazul în care `lhs < rhs`
  - `0` — în cazul în care `lhs = rhs`
  - număr pozitiv — în cazul în care `lhs > rhs`
- numărul returnat reprezintă diferența caracterelor de la prima poziție unde ele diferă în cele două string-uri
- Erori posibile
  - `lhs` sau `rhs` nu sunt string-uri valide

Funcția strtok

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

- Desparte în bucăți (token-uri) string-ul pe baza delimitatorilor
- Fiecare caracter din `str` care este un delimitator este înlocuit cu caracterul nul - șirul de intrare se modifică
- Parametrii
  - `char* src` - șirul de caractere sursă
  - `const char* delim` - delimitatorii
- Valoare returnată
  - Un pointer la următoarea bucată
  - Pointerul `NULL` dacă nu există următoarea bucată
- Prima dată se apelează cu string-ul de despărțit apoi cu str = `NULL` pentru a sări la următoarea bucată
- Erori posibile
  - `str` nu este un string valid

Parsare string-uri - exemplu simplu

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

int main(){
   char s[] = "sir de cuvinte, separate..,.prin.diferite caractere";
   char* delim = " ,.";
   char* tok = strtok(s, delim);
   while(tok != NULL){
       puts(tok);
       tok = strtok(NULL, delim);
   }
   return 0;
}
```

- includem biblioteca string.h
- *s* este declarat ca tablou fiindcă se va modifica

- delimitatori
- primul apel la strtok cu string-ul *s*
- cât timp este alt cuvânt
- afișăm cuvântul - se oprește la caracterul 0 introdus
- la apelurile ulterioare se trimite pointerul `NULL` în locul lui *s*

- De multe ori se poate evita folosirea lui `strtok` care modifică șirul original

Funcții din string.h

- memcpy - copiază o zonă de memorie
- memmove - mută o zonă de memorie
- memcmp - compară două zone de memorie
- memchr - caută un byte într-o zonă de memorie
- memset - setează toți bytes dintr-o zonă la o valoare

https://en.cppreference.com/w/c/string/byte

Funcția memmove

```
void* memmove(void* dest, const void* src, size_t count);
```

- Mută conținutul de memorie de mărime `count` bytes de la sursa `src` la destinația `dest`
- Parametrii
  - `char* dest` - pointer la zona de destinație
  - `const char* src` - pointer la zona de sursă
  - `size_t count` - mărimea zonei în bytes (octeți)
- Valoare returnată
  - O copie a pointerului `dest` în caz de succes
  - În caz de eșec se setează `dest[0]` la 0 dacă `dest` nu este pointerul NULL
- Este permis ca cele două zone să se suprapună
- Erori posibile
  - `dest` nu este pointer valid
  - zona alocată de la `dest` nu este suficient de mare

Trimitere string-uri la funcții

- Se respectă regulile și recomandările prezentate la trimiterea tablourilor la funcții
- String-urile se trimit prin valoare
  - se trimite o copie a pointerului la primul caracter
  - modificarea conținutului din string se reflectă și în exteriorul funcției
  - modificarea pointerului nu afectează pointerul original (este trimis prin valoare)
  - nu se poate calcula memoria ocupată cu `sizeof` fiindcă se trimite ca pointer nu ca tablou

Alocare dinamică string-uri

- Se respectă regulile și recomandările prezentate la alocare dinamică
- Un string se alocă dinamic dacă
  - este de dimensiune mare - tipic mai mare ca 1000 caractere
  - este returnat de la o funcție - pentru a rămâne valabil
    - trebuie dezalocat după folosire
  - își schimbă dimensiunea în timpul programului
  - nu are dimensiune cunoscută la începutul programului
    - string-uri de dimensiune redusă se pot aloca pe stivă cu dimensiune variabilă cu `char s[n];`

Șir de string-uri

- Se pot stoca ca un tablou bidimensional de caractere alocat static `char s[dim1][dim2]`
  - toate string-urile vor avea alocate același număr de bytes pentru lungime (`dim2`)
  - nu se pot schimba pointerii - trebuie să folosim `strcpy`
  - recomandat dacă șirurile au dimensiuni limitate
- Se pot stoca ca un tablou bidimensional de caractere alocat dinamic `char** s`
  - string-urile pot avea lungimi diferite
  - se pot schimba direct pointerii
  - flexibil

Șir de string-uri - exemplu complex

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

int main(){
    char** cuvinte;
    char s[100];
    int n;
    puts("Cate cuvinte?");
    scanf("%d", &n);
    cuvinte = calloc(n, sizeof(char*));
    for(int i=0; i<n; i++){
        scanf("%s", s);
        cuvinte[i] = calloc(strlen(s)+1, sizeof(char));
        strcpy(cuvinte[i], s);
    }

    //... continuat pe slide-ul urmator
```

- citește n cuvinte și le sortează alfabetic
- folosim alocare dinamică pentru a stoca un șir de string-uri

- în s citim cuvintele și se le mutăm în `cuvinte`
- atenție la alocarea spațiului și pentru caracterul nul

Șir de string-uri - exemplu complex

```
    //...main
    for(int i=0; i<n; i++){
        for(int j=i+1; j<n; j++){
            if (strcmp(cuvinte[i], cuvinte[j])>0){
                char* tmp = cuvinte[i];
                cuvinte[i] = cuvinte[j];
                cuvinte[j] = tmp;
            }
        }
    }
    for(int i=0; i<n; i++)
        puts(cuvinte[i]);
    return 0;
}
```

- sortare prin selection sort pe baza lui `strcmp`
- mutăm cel mai mic element pe prima poziție, sortăm restul la fel
- timp pătratic în *n*
- în acest caz este suficient să interschimbăm pointerii
- nu este nevoie de copierea șirurilor
- afișare cuvinte sortate

Analiză problemă - căutare subșir

- Se cere determinarea numărului de apariții ale string-ului *t* în string-ul *s*
  - aparițiile pot să se suprapună
  - exemplu, pentru `s = "xabacababa", t = "aba"`
    - apare de 3 ori începând de la pozițiile subliniate
- Notăm cu *m* lungimea lui *t*, cu *n* lungimea lui *s*
- Algoritmul naiv
  - pentru fiecare poziție posibilă din *s*
  - se verifică dacă substring-ul este egal cu *t*
  - se compară caracter cu caracter
  - *(n-m) * m* comparații
    - produsul poate fi mare dacă *m* și *n* sunt peste 1000

Analiză problemă - căutare subșir

- Algoritmul Karp-Rabin
  - pentru fiecare string sau substring se calculează o valoare întreagă care îl va reprezenta
    - funcție de dispersie = eng. hash function

Analiză problemă - căutare subșir

- Algoritmul Karp-Rabin
  - pentru fiecare string sau substring se calculează o valoare întreagă care îl va reprezenta
    - funcție de dispersie = eng. hash function
  - fiindcă sunt mult mai multe string-uri decât valorile întregi posibile pentru un tip (de ex. `int`) vom avea cazuri când două string-uri diferite se mapează la aceeași valoare
  - încercăm să proiectăm o funcție care să reducă riscul coliziunilor
  - aplicăm funcția pe *t* și pe un subșir din *s*
    - dacă valorile rezultante sunt egale este o șansă ca cele două string-uri sunt egale - nu este sigur
      - trebuie să efectuăm comparația caracter cu caracter
    - dacă valorile rezultante nu sunt egale atunci sigur cele două string-uri nu sunt egale
      - nu este nevoie de comparație

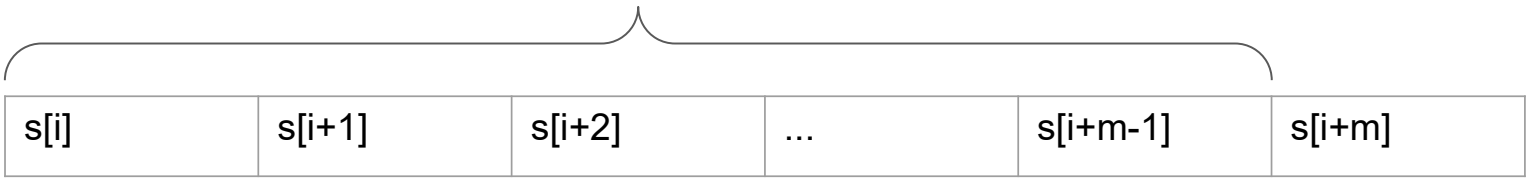
Analiză problemă - căutare subșir

- Algoritmul Karp-Rabin
  - se definește funcția de dispersie pe un string (*p* e un număr prim)
  - $h(t) = t[0] * p^{m-1} + t[1] * p^{m-2} + \ldots t[m-1]\ p^0$
  - se aplică funcția pe stringul *t* și salvăm
  - *ht = h(t)*
  - se aplică funcția pe primele *m* caractere ale string-ului *s*
  - *hs = h(s[0 : m-1])*
  - *nr = 0*
  - pentru fiecare poziție *i* de la 0 la *n-m*
    - dacă *ht = hs*
      - se compară *t* cu subșirul *s[i:i+m-1]* caracter cu caracter pentru a determina dacă avem apariție sau nu
      - dacă sunt egale creștem *nr*
    - dacă *i < n-m*
      - calculăm funcția de dispersie pentru următorul subșir
      - *hs = next(hs, s, i)*

Analiză problemă - căutare subșir

- Algoritmul Karp-Rabin
  - *hs = next(hs, s, i)*
  - Având funcția de dispersie pentru un substring cum calculăm eficient funcția pentru substring-ul de la poziția următoare?
  - notăm cu hs_i funcția de dispersie pentru substringul care începe de la poziția *i*
    - dorim să obținem hs_{i+1} din hs_i

$hs_i = s[i] * p^{m-1} + s[i+1] * p^{m-2} + \ldots + s[i+m-2] * p^1 + s[i+m-1] * p^0$



$hs_{i+1} = s[i+1] * p^{m-1} + s[i+2] * p^{m-2} + \ldots + s[i+m-1] * p^1 + s[i+m] * p^0$

Analiză problemă - căutare subșir

- Algoritmul Karp-Rabin
  - *hs = next(hs, s, i)*
  - Observăm că
    - $hs_{i+1} = hs_i * p - s[i] * p^m + s[i+m] * p^0$
    - se poate precalcula și salva p^m

$hs_i = s[i] * p^{m-1} + s[i+1] * p^{m-2} + \ldots + s[i+m-2] * p^1 + s[i+m-1] * p^0$

$hs_{i+1} = s[i+1] * p^{m-1} + s[i+2] * p^{m-2} + \ldots + s[i+m-1] * p^1 + s[i+m] * p^0$

Algoritmul Karp-Rabin - implementare 1

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

const long long p = 17;
long long hash(char* s, int m){
   long long res = 0;
   for(int i=0; s[i] && i<m; i++)
       res = res * p + s[i];
   return res;
}

int main(){
   char t[] = "bra";
   char s[] = "Abra abracadabra\n"
              "I wanna reach out and grab ya\n"
              "Abracadabra\n"
              "Abracadabra";

    //... continuat pe slide-ul urmator
```

- se alege un număr prim
- funcția de dispersie calculată pe primele *m* caractere din *s*
- metoda Horner pentru evaluarea unui polinom

- se definește string-ul *s* pe mai multe linii
- linie nouă în cod sursă nu înseamnă linie nouă în string - aceasta trebuie introdusă explicit

Algoritmul Karp-Rabin - implementare 2

```
//...main
int m = strlen(t);
int n = strlen(s);
long long ht = hash(t, m);
long long hs = hash(s, m);
int nr = 0;
long long pm = 1;
for(int i=0; i<m; i++)
    pm *= p;
for(int i=0; i<=n-m; i++){
    if (ht == hs){
        int match = 1;
        for(int j=0; j<m && match; j++)
            if (s[i+j]!=t[j])
                match = 0;
         nr += match;
    }
    if (i+m < n)
        hs = hs * p - pm * s[i] + s[i+m];
}
printf("%d\n", nr);
return 0;
}
```

```
char t[] = "bra";
char s[] = "Abra abracadabra\n"
           "I wanna reach out and grab ya\n"
           "Abracadabra\n"
           "Abracadabra";
```

- precalculăm p^m
- de ce nu folosim pow?

- în caz de egalitate verificăm caracter cu caracter

- dacă match este 1 se poate procesa apariția - de exemplu putem afișa pozițiile de început
- pentru ultima poziție validă nu trebuie să calculăm următoarea valoare a funcției de dispersie

Analiză problemă - căutare subșir

- Algoritmul Karp-Rabin
  - în general este mai eficient decât algoritmul naiv
  - dacă nu avem coliziuni deloc atunci necesită *n* + *m* * (număr de apariții) comparări
  - este posibil fiindcă funcția de dispersie se poate modifica ușor
    - se calculează prin glisare - eng. rolling hash
  - putem ignora overflow în calcularea funcției de dispersie
  - dacă sunt multe potriviri atunci nu este mai eficient
  - există alți algoritmi cu timp bun
    - Knuth-Morris-Pratt
    - Aho–Corasick