Ministerul Educaţiei și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică şi Microelectronică

**RAPORT**

Lucrare de laborator Nr.6

*la Programarea Calculatoarelor*

Tema: Prelucrarea caracterelor și a șirurilor de caractere

A efectuat: st. gr. SI-212 Șeremet Alexandru

A verificat: lect. asist. Mititelu Vitalii

Chişinău 2021

**Scopul:** Programarea algoritmilor de prelucrare a caracterelor și a șirurilor de caractere prin utilizarea funcțiilor predefinite de prelucrare a caracterelor și a șirurilor de caractere în limbajul C.

**Sarcina:** Scrieți un program care citește de la tastatură o propoziție, apoi efectuează calculele indicate în variantă și afișează pe ecran rezultatul:

**Varianta 15:**

Să se elimine din propoziție cuvintele identice.

**Rezumat:**

În C++ există mai multe modalități de a reprezenta șirurile de caractere. În acest articol vom discuta despre șirurile de caractere reprezentate ca tablouri unidimensionale cu elemente de tip char, reprezentare care provine din limbajul C.

Aceste șiruri se mai numesc **null-terminated byte string (NTBS)**. În reprezentarea internă, după ultimul caracter (byte, octet) valid din șir se află caracterul '\0' – caracterul cu codul ASCII 0, numit și **caracter nul**.

Astfel, pentru reprezentarea în C/C++ a cuvântului copil, care are 5 caractere, se vor folosi 6 octeți, cu valorile: 'c', 'o', 'p', 'i', 'l', '\0'.

## Declararea unui șir de caractere

Un șir de caractere se declară în C++ astfel:

char s[11];

S-a declarat un șir care poate memora maxim 11 caractere, cu indici 0 1 ... 10. Șirul s poate memora cel mult 10 caractere utile, după ultimul caracter util fiind memorat caracterul '\0'.

De asemenea, la declararea unui șir acesta poate fi inițializat. Următoarele exemple declară șiruri de caractere și le inițializează cu șirul "copil":

char s[11] = "copil"; // se folosesc doar 6 caractere

char t[]="copil"; // se aloca automat 6 octeti pentru sirul t: cele 5 litere si caracterul nul \0

char x[6]={'c','o','p','i','l','\0'}; // initializarea este similara cu cea a unui tablou oarecare - sirurile de caractere sunt tablouri

char z[]={'c','o','p','i','l','\0'}; // se aloca automat 6 octeti pentru sir

## Tipul char \*. Legătura dintre pointer-i și tablouri

Considerăm declarația:

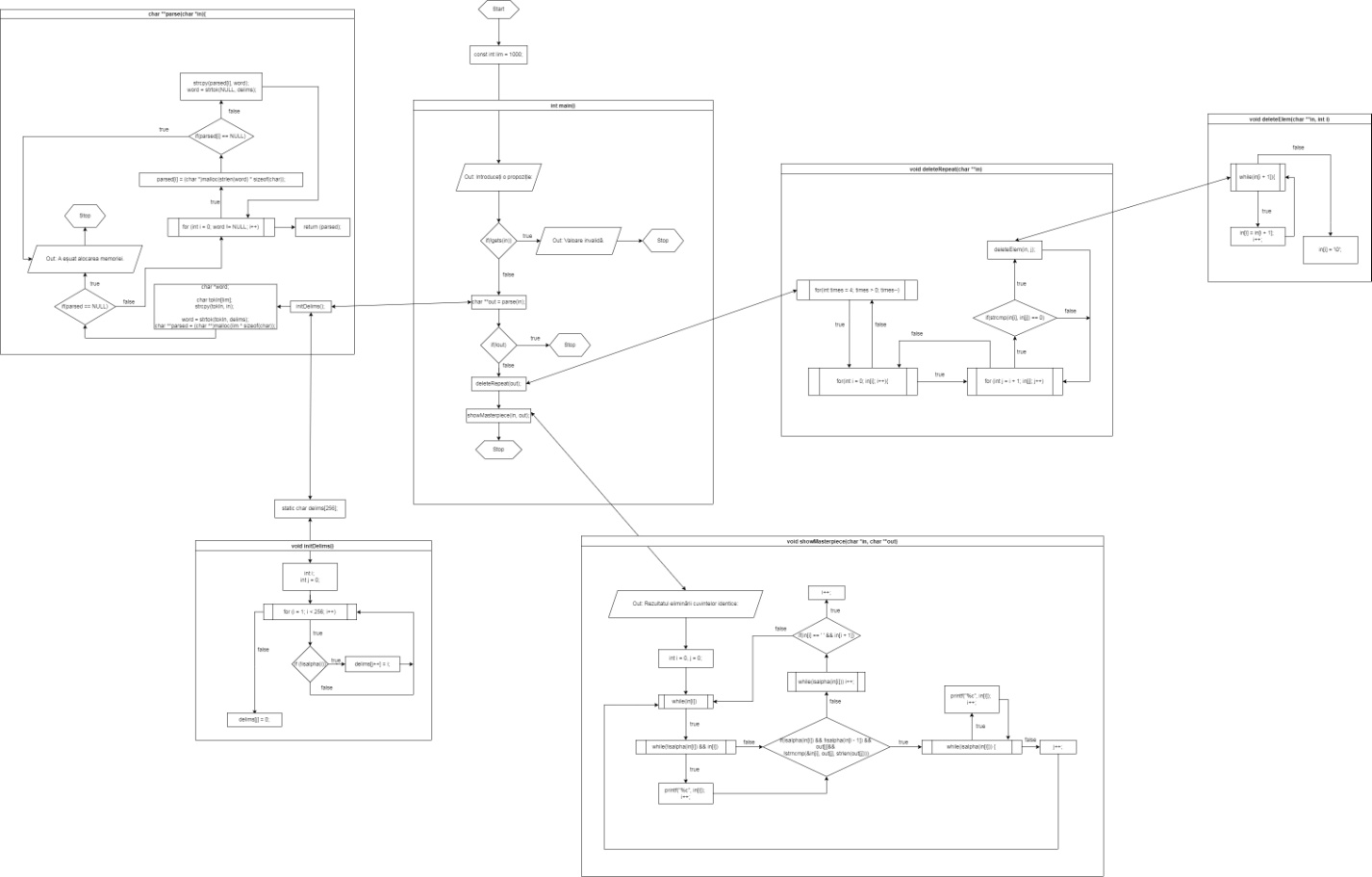
char \* p , s[31] = "pbinfo";

Ce este p? Este un pointer la char, adică o variabilă a cărei valoare este adresa unei date de tip char. Întrebarea este a cărei date? În acest moment a niciuneia, deoarece variabila p nu a fost inițializată, iar valoare ei este o adresă aleatorie. Șansele ca ea să reprezinte adresa unei date de tip char din spațiul de memorie al programului nostru sunt la fel de mici ca șansele ca valoarea inițială a unei variabile de tip int să fie în intervalul 1 ... 100.

Ce este s? Spunem că este un șir de caractere, dar practic s este tot un pointer. Valoarea sa este adresa primului element din șir, adică adresa lui s[0]. Observăm că de fapt, variabilele p și s sunt de același tip, pointer la char. Diferența dintre cele două variabile este că s memorează o adresa de memorie unde începe un șir de caractere (la acea adresă există o dată de tip char ) în timp ce p memorează o adresă aleatorie.

Cu ce putem inițializa pointer-ul p? Cu adresa unei date de tip char. O asemenea dată este orice element al unui șir de caractere, de exemplu orice element din s. Dacă p reprezinta adresa unui caracter dintr-un șir, atunci cu p se pot face toate operațiile care se pot face cu acel șir.

**Schema bloc a algoritmului de soluționare a problemei:**



**Codul deplin al programului:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

const int lim = 1000;

void deleteElem(char \*\*in, int i) {

    while(in[i + 1]){

        in[i] = in[i + 1];

        i++;

    }

    in[i] = '\0';

}

void deleteRepeat(char \*\*in) {

    for(int times = 4; times > 0; times--){

        for(int i = 0; in[i]; i++){

            for (int j = i + 1; in[j]; j++) {

                if(strcmp(in[i], in[j]) == 0) {

                    deleteElem(in, j);

                }

            }

        }

    }

}

static char delims[256];//valoare arbitrara

void initDelims() {

    int i;

    int j = 0;

    for (i = 1; i < 256; i++) {

        if (!isalpha(i))

            delims[j++] = i;

    }

    delims[j] = 0;

}

char \*\*parse(char \*in){

    initDelims();

    char \*word;

    char tokIn[lim];

    strcpy(tokIn, in);

    word = strtok(tokIn, delims);

    char \*\*parsed = (char \*\*)malloc(lim \* sizeof(char));

    if(parsed == NULL) {

        printf("A eșuat alocarea memoriei.\n");

        return NULL;

    }

    for (int i = 0; word != NULL; i++){

        parsed[i] = (char \*)malloc(strlen(word) \* sizeof(char));

        if(parsed[i] == NULL) {

            printf("A eșuat alocarea memoriei.\n");

            return NULL;

        }

        strcpy(parsed[i], word);

        word = strtok(NULL, delims);

    }

    return (parsed);

}

void showMasterpiece(char \*in, char \*\*out) {

    printf("\nRezultatul eliminării cuvintelor identice:\n\n");

    int i = 0, j = 0;

    while(in[i]) {

        while(!isalpha(in[i]) && in[i]) {

            printf("%c", in[i]);

            i++;

        }

        if(isalpha(in[i]) && !isalpha(in[i - 1]) &&

            out[j] &&

            !strncmp(&in[i], out[j], strlen(out[j]))) {

            while(isalpha(in[i])) {

                printf("%c", in[i]);

                i++;

            }

            j++;

        }

        else {

            while(isalpha(in[i])) i++;

            if(in[i] == ' ' && in[i + 1])

                i++;

        }

    }

    printf("\n");

}

int main () {

    char in[lim];

    printf("\n-Pentru programul dat, un cuvânt este\no succesiune de litere care nu e\ndelimitată de nimic.\n\n");

    printf("-Cuvintele la care majusculele și minusculele\nnu coincid nu se vor considera identice.\n\n");

    printf("Introduceți o propoziție:\n\n");

    if(!gets(in)) {

        printf("Valoare invalidă.");

        return(1);

    }

    char \*\*out = parse(in);

    if(!out){

        return (1);

    }

    deleteRepeat(out);

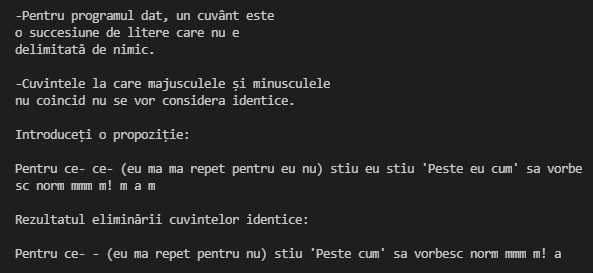
    showMasterpiece(in, out);

    return (0);

}

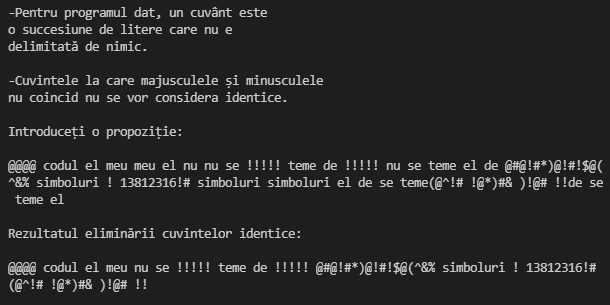
**Execuția programului:**

1. Set de date nr.1 : „Pentru ce- ce- (eu ma ma repet pentru eu nu) stiu eu stiu 'Peste eu cum' sa vorbesc norm mmm m! m a m”



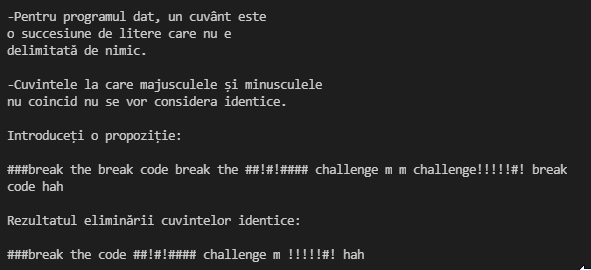
Sarcina de bază este îndeplinită cu succes. Pe lângă asta, simbolurile sunt luate în considerare și puse adecvat.

1. Set de date nr.2: „@@@@ codul el meu meu el nu nu se !!!!! teme de !!!!! nu se teme el de @#@!#\*)@!#!$@(^&% simboluri ! 13812316!# simboluri simboluri el de se teme(@^!# !@\*)#& )!@# !!de se teme el”



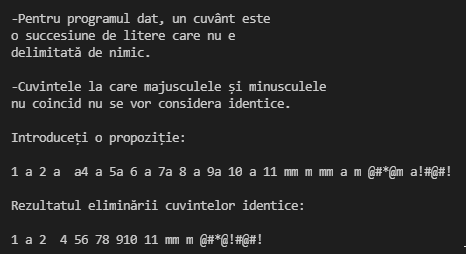
După cum se vede, simbolurile și caracterele alfabetice se pot afla în orice succesiune fără devieri de la răspunsul corect.

1. Set de date nr.3: „###break the break code break the ##!#!#### challenge m m challenge!!!!!#! break code hah”



Simbolurile și numerele sunt pur și simplu ignorate, deoarece ele nu constituie un cuvânt.

1. Set de date nr.4: „1 a 2 a a4 a 5a 6 a 7a 8 a 9a 10 a 11 mm m mm a m @#\*@m a!#@#!”



Indiferent de numărul de repetiții, codul le șterge pe toate.

**Concluzii:**

1. Șirurile de caractere sunt, esențial, un pointer la un caracter care desemnează începutul lui, celelalte caractere ce urmează, și simbolul ‘\0’ care indică sfârșitul șirului.
2. Este foarte important să cunoști cum lucrează memoria și cum să o aloci și să o manipulezi în dependență de necesitate, deoarece integritatea șirurilor de caractere depinde de succesivitatea adreselor lor în memorie.
3. Pentru a ușura lucrul cu șirurile de caractere, limbajul C oferă o bibliotecă dedicată lor - <string.h> - care include funcții, precum strtok(), strcat(), strcpy(), strcmp() etc. precum și unele versiuni cu o limită de iterații: strncat(), strncpy, strncmp() etc.

**Surse:**

* **pbinfo.ro, Șiruri de caractere în C++:**

https://www.pbinfo.ro/articole/19/siruri-de-caractere-in-cpp