Link pagina prelucrare siruri

+ pt olimpiada [Testarea timpului de executie a unui program]

+plus pointeri la caractere pt olimpiada

Exercitii:

* Funcţia cin.get() utilizează o zonă intermediară de memorie (buffer) astfel:
  + Se memorează în buffer caracterele care s-au tastat până la apăsarea tastei ENTER.
  + Se analizează pe rând primele nr-1 caractere (dacă există nr-1 caractere) din buffer testându-se dacă printre aceste caractere se află caracterul c. Dacă se întâlneşte caracterul c, atunci, toate caracterele tastate înaintea lui c vor fi adaugate şirului şi vor fi eliminate din buffer aceste caractere şi caracterul c din buffer.
* Din acest motiv, folosirea repetată a funcţiei cin.get () poate funcţiona eronat. Motivul este că, după executarea unei citiri, este posibil ca bufferul să nu se golească complet.
* Se va utiliza :
* cin.get (s1, 100);
* cin.get(); // se mai citeşte un caracter din buffer – cu scopul de a elimina acest caracter cin.get(s2, 100);

Transformare litera mica in litera mare: c = c – ‘a’ + ‘A’

Transformare din caracterul cifra in valoarea cifrei (exemplu: ‘7’ in 7): c = c – ‘0’

|  |
| --- |
| cout<<s; *// scrie "ana are mere."*  cout<<s+1; *// scrie "na are mere."* |

*strcat(v,s); // concateneaza sirul s in v (adauga ce este in s la sfarsitul lui v)*

*strcpy(v + strlen(v) , s); // echivalent cu strcat.*

#include<iostream.h>  
#include<string.h>  
  
void main()  
{char \*p,text[300],sep[]=" .,;!?",cuvinte[20][100];  
int i,k=0;  
cout<<"Introduceti propozitia dorita:"<<endl;  
cin.getline(text,300);  
p=strtok(text,sep);  
while(p!=0) {k++;  
                 strcpy(cuvinte[k],p);  
                 p=strtok(NULL,sep);  
                 }  
for(i=1;i<=k;i++) cout<<cuvinte[i]<<endl;  
}

In continuare vom numi caractere albe ca fiind acele care afisate pe ecran nu au un rezultat vizibil (spatiu, enter, tab, etc).

Exercitii: Stergerea caracterului de pe pozitia k din sir: Deoarece se recomanda ca sirurile care sunt date ca argument functiei strcpy sa nu se suprapuna in memorie, folosim o variabila auxiliara. strcpy(aux, sir+k+1) strcpy(sir+k, aux) Numararea tuturor aparitiilor literei ‘n’ intr-un sir de caractere char\* aparitie = strchr(sir1, ‘n’); int nr\_aparitii = 0; while (aparitie!=NULL){ nr\_aparitii++; aparitie = strchr(aparitie+1, ‘n’);

Ex: Sa se afiseze toate pozitiile unui caracter intr-un sir

#include <iostream.h>

#include <string.h>

void main()

{char a[100],\*p,c;

cin.get(a,100);

cin>>c;

p=strchr(a,c);

while (p)

{cout<<"Pozitia "<<p-a<<endl;

 p++;

 p=strchr(p,c);}}

Numararea tuturor aparitiilor literei ‘n’ intr-un sir de caractere char\* aparitie = strchr(sir1, ‘n’); int nr\_aparitii = 0; while (aparitie!=NULL){ nr\_aparitii++; aparitie = strchr(aparitie+1, ‘n’);

Funcția C++ strcpy copiază un șir de caractere sursă în alt șir de caractere, destinație. Sintaxa ei este:

[view source](https://www.pbinfo.ro/?pagina=intrebari-afisare&id=30#viewSource)

[print](https://www.pbinfo.ro/?pagina=intrebari-afisare&id=30#printSource)[?](https://www.pbinfo.ro/?pagina=intrebari-afisare&id=30#about)

1.char \* strcpy(char \* dest, const char \* src );

În [documentația C++](http://en.cppreference.com/w/c/string/byte/strcpy) este precizat că această funcție are un comportament impredictibil dacă șirurile src și dest se suprapun.

Cu toate acestea, a devenit o practică uzuală ștergerea unor caractere dintr-un șir cu un apel de forma: strcpy(s + i, s + i + 1); (pentru ștergerea caracterului de indice i din șirul s). Numeroase soluții ale problemelor cu șiruri de caractere de pe [pbInfo.ro](https://www.pbinfo.ro/), inclusiv soluții oficiale, folosesc această metodă pentru eliminarea unor caractere dintr-un șir, metodă care a funcționat în versiunile mai vechi ale compilatorului GCC, dar nu mai funcționează în versiunea curentă.

Recomandăm tuturor utilizatorilor [pbInfo.ro](https://www.pbinfo.ro/) să evite eliminarea caracterelor dintr-un șir prin metoda descrisă mai sus. O metodă alternativă, care funcționează corect, utilizează un șir de caractere suplimentar, astfel:

[view source](https://www.pbinfo.ro/?pagina=intrebari-afisare&id=30#viewSource)

[print](https://www.pbinfo.ro/?pagina=intrebari-afisare&id=30#printSource)[?](https://www.pbinfo.ro/?pagina=intrebari-afisare&id=30#about)

1.char s[256], t[256];

2....

3.//eliminam caracterul de indice i din s

4.strcpy(t, s + i + 1);

5.strcpy(s + i, t);

Scrieti programul care citeste de la tastatura un sir de caractere al alfabetului englez si afiseaza pe randuri diferite sirul de caractere dat cat si toate prefixele acestuia de lumgime cel putin 1, in ordinea descrescatoare a lungimii prefixelor, aliniate la stanga

Scrieti programul care citeste de la tastatura un sir de caractere al alfabetului englez si afiseaza sirul de caractere dat precum si toate sufixele acestuia de lumgime cel putin 1, fiecare pe cate o linie in ordinea crescatoare a lungimii sufixelor, aliniate la stanga.

Se consideră un text cu cel mult 70 de caractere (litere mici ale alfabetului englez si spatii), în care cuvintele sunt separate prin unul sau mai multe spatii. Înaintea primului cuvânt si după ultimul cuvânt nu există spatii. Scrieti un program C/C++ care citeste de la tastatură un text de tipul mentionat mai sus si afisează pe ecran numărul de cuvinte în care apare litera a

Se citeste un sir. Sa se preia cuv intr un vector de cuvinte (si nr lor) utilizand functia strtok.

Sa se inverseze cuv dintr un sir (apoi sa apara in ordine inversa in sir).

Ce face      Functia **stricmp** ?  
int stricmp(sir1,sir2); – are acelasi rol cu strcmp, cu deosebirea ca *nu face distinctie* intre literele mari si cele mici ale alfabetului (i = ignore).

Functia **strspn** cu forma generala  
int strspn(sir1,sir2); – are rolul de a returna numarul de caractere ale sirului sir1 (caractere consecutive care incep obligatoriu cu primul caracter) care se gasesc in sirul sir2.

*Exemplu*:  
strspn(“AB2def”,”1B3AQW”);  returneaza 2, pentru ca primele 2 caractere ‘A’ si ‘B’

din sir1 se gasesc in sir2.

strspn(“FAB2def”,”16A32BF”);  returneaza 0, deoarece caracterul ‘F’ cu care incepe sir1 nu se gaseste in sir2.

       Functia **strcspn** cu forma generala  
int strspn(sir1,sir2); – are rolul de a returna numarul de caractere ale sirului sir1 (caractere consecutive care incep obligatoriu cu primul caracter) care *nu* se gasesc in sirul sir2.

*Exemplu*:  
strspn(“AB2def”,”123”);  returneaza 2, pentru ca primele 2 caractere din sir1 nu se gasesc in sir2.

//Se citeste un sir de caractere care nu contine caractere albe. Sa se decida daca sirul este alcatuit exclusiv din caractere numerice.

#include <iostream.h>

#include <conio.h>

#include <string.h>

void main()

{char text[100],cifre[]="0123456789";

clrscr();

cout<<"Dati sirul:";cin.get(text,100);

if (strcspn(cifre,text)==strlen(text))

cout<<"exclusiv numeric";

else cout<<”nenumeric”;

getch();}

Sa se determine daca textul are cuvinte distincte (se ignora diferenta de cheie).

Functia **strbrk** cu forma generala  
strpbrk(sir1,sir2); – actioneaza in felul urmator:

o        Cauta primul caracter al sirului sir1 in sir2. Daca este gasit, returneaza adresa sa din cadrul sirului sir1 si executia se termina. Altfel, se trece la pasul urmator.

o        Cauta al doilea caracter al sirului sir1 in sir2. Daca este gasit, returneaza adresa sa din cadrul sirului sir1 si executia se termina. Altfel, se trece la pasul urmator.

o        …

o        Daca nici un caracter al sirului sir1 nu apartine sirului sir2, functia returneaza adresa nula.

# Se introduc de la tastatura cuvinte pana la citirea cuvantului stop.

# 15. Se citeste un numar real de la tastatura. Sa se trunchieze astfel incat cifrele ramase sa formeze o secventa monotona. Ex. 34. 59483 ®34.59; 24.1276 ® 24

Se dă un șir de 255 caractere, format din litere mici și spații. Să se determine câte vocale din

șir sunt cuprinse între două consoane.

Se citesc de la tastatură două propoziții terminate cu ENTER. În aceste propoziții cuvintele

sunt despărțite prin unul sau mai multe spații. Să se afișeze propoziția care are mai multe

cuvinte. Câte cuvinte are această propoziție?

4. Se citește de la tastatură o propoziție terminată cu PUNCT. Cuvintele ce formează propoziția

sunt despărțite prin unul sau mai multe spatii. Să se afișeze fiecare cuvânt din propoziție pe

cate un rând și în dreapta fiecărui cuvânt inversul cuvântului respectiv.

5. Se citesc de la tastatură două cuvinte. Să se afișeze literele comune celor două cuvinte. Dacă

o literă apare de mai multe ori, atunci se va afișa o singură dată.

5. Subiect tip II - Varianta 024 problema 5-Bacalaureat 2009

Scrieţi un program C/C++ care citeşte de la tastatură două şiruri de caractere formate din maximum 100 litere mici ale alfabetului englez şi afişează pe ecran cel mai lung sufix comun al celor două şiruri de caractere. Dacă cele două şiruri nu au niciun sufix comun, atunci programul va afişa pe ecran mesajul NU EXISTA.

**Exemplu:** pentru şirurile **marina** şi **elena** se va afişa **na**

6. Subiect tip II - Varianta 028 problema 5-Bacalaureat 2009

Şirul de caractere **s2** este “**clona**” şirului de caractere **s1** dacă se poate obţine din **s1** prin eliminarea tuturor apariţiilor unei singure vocale. Se consideră vocală orice literă din mulţimea {a,e,i,o,u}. Scrieţi programul C/C++ care citeşte de la tastatură un cuvânt format din cel mult 20 litere mici ale alfabetului englez şi **afişează pe ecran** (dacă există), **toate “clonele” acestui cuvânt**, fiecare pe câte o linie a ecranului.

**Exemplu**: pentru cuvântul **informatica** se afişează, nu neapărat în această ordine, “**clonele**” scrise alăturat.

**nformatca**

**infrmatica**

**informtic**

9. Subiect tip II - Varianta 039 problema 5-Bacalaureat 2009

Un şir cu maximum **255 de caractere** conţine cuvinte separate prin câte un spaţiu. Cuvintele sunt formate numai din litere mici ale alfabetului englez. Scrieţi un program C/C++ care citeşte de la tastatură un astfel de şir şi îl afişează pe ecran modificat, **inversând prin**

**oglindire doar cuvintele care încep cu vocală**, ca în exemplu. Se consideră ca fiind vocale următoarele litere: a, e, i, o, u.

**Exemplu**: pentru şirul: **maine este proba la informatica** se va afişa: **maine etse proba la acitamrofni**

10. Subiect tip II - Varianta 040 problema 5-Bacalaureat 2009

Un şir cu maximum **255 de caractere** conţine cuvinte cuvinte formate numai din litere mici ale alfabetului englez. Fiecare cuvânt este urmat de un caracter \*. Scrieţi un program C/C++ care citeşte un astfel de şir şi afişează pe ecran **şirul obţinut prin eliminarea tuturor apariţiilor primului cuvânt**, ca în exemplu.

**Exemplu**: pentru şirul: **bine\*albine\*foarte\*bine\*** se va afişa: **\*albine\*foarte\*\***

12. Subiect tip II - Varianta 045 problema 5-Bacalaureat 2009

Scrieţi programul C/C++ care citeşte de la tastatură un şir de cel mult **40 de caractere**, format doar din litere ale alfabetului englez, şi care **afişează pe ecran toate şirurile obţinute prin eliminarea succesivă a câte unei singure litere din şirul citit**, ca în exemplu. Şirurile se vor afişa câte unul pe câte o linie a ecranului.

**Exemplu:** dacă se citeşte şirul **abbc** atunci pe ecran se va afişa:

**bbc**

**abc**

**abc**

**abb**

13. Subiect tip II - Varianta 046 problema 5-Bacalaureat 2009

Se consideră un text format doar din spaţii şi litere mici ale alfabetului englez, care începe cu o literă şi care conţine cel puţin o vocală din multimea {a,e,i,o,u}. Scrieţi programul C/C++ care citeşte de la tastatură un şir cu cel mult **100 de caractere**, ca cel descris mai sus şi care determină **transformarea acestuia prin înlocuirea fiecărei vocale din text cu litera imediat următoare din alfabet** (a se înlocuieşte cu b, e se înlocuieşte cu f ş.a.m.d.). Programul va afişa pe ecran şirul obţinut.

**Exemplu:** dacă şirul citit este examen de **bacalaureat**, după modificare se afişează: **bbcblbvrfbt**

#include

#include

using namespace std;

int main()

{

char s[256], \*p;

cin.get(s,255);

p=strtok(s," ");

while (p)

{strrev(p);

cout<<p<<endl;

p=strtok(NULL, " ") ;

}

return 0;

}

Testarea timpului de execuție al unui program

* [1Testarea duratei de execuție a unui program](http://algopedia.ro/wiki/index.php/Testarea_timpului_de_execu%C8%9Bie_al_unui_program#Testarea_duratei_de_execu.C8.9Bie_a_unui_program)
* [2Execuție a unui program pînă la expirarea timpului alocat](http://algopedia.ro/wiki/index.php/Testarea_timpului_de_execu%C8%9Bie_al_unui_program#Execu.C8.9Bie_a_unui_program_p.C3.AEn.C4.83_la_expirarea_timpului_alocat)
  + [2.1Exemplul unu](http://algopedia.ro/wiki/index.php/Testarea_timpului_de_execu%C8%9Bie_al_unui_program#Exemplul_unu)
  + [2.2Exemplul doi](http://algopedia.ro/wiki/index.php/Testarea_timpului_de_execu%C8%9Bie_al_unui_program#Exemplul_doi)
* [3Concluzie](http://algopedia.ro/wiki/index.php/Testarea_timpului_de_execu%C8%9Bie_al_unui_program#Concluzie)
  + [3.1Efectul observatorului](http://algopedia.ro/wiki/index.php/Testarea_timpului_de_execu%C8%9Bie_al_unui_program#Efectul_observatorului)

Testarea duratei de execuție a unui program

Pentru a testa cît timp rulează programul nostru vom folosi tot funcția gettimeofday().

#include <stdio.h>

#include <sys/time.h> // pentru structura timeval

long long start; // momentul de timp cind a pornit programul (in microsecunde)

// returneaza timpul in microsecunde (o milionime de secunda)

inline long long getTime() {

struct timeval tv;

gettimeofday(&tv, NULL);

return tv.tv\_sec \* 1000000LL + tv.tv\_usec;

}

int main() {

int i, suma;

long long durata;

start = getTime(); // timpul de inceput al programului, prima instructiune din main()

// ... urmeaza programul propriu-zis

// ... toate calculele se fac aici

// ... iata cod care "consuma" timp

suma = 0;

for ( i = 0; i < 1000000; i++ )

suma = (suma + i) % 1234;

// afisam suma, altfel codul de mai sus este eliminat de compilator

printf( "%d\n", suma );

// ... am terminat codul consumator de timp

// ... apoi, la final de program, chiar inainte de return 0

// ... calculam timpul de executie

durata = getTime() - start;

printf( "Timpul de executie al programului in microsecunde: %lld\n", durata );

return 0;

}

Execuție a unui program pînă la expirarea timpului alocat

Pentru a executa un program pînă la un moment dat vom folosi aceeași funcție gettimeofday(). Acest lucru ne folosește în unele situații precum:

* atunci cînd programul nostru calculează un rezultat care poate fi îmbunătățit în timp (gen mutarea în timpul unui joc)
* atunci cînd la olimpiadă scriem un program ce folosește backtracking pentru a găsi soluția, care este posibil să nu se încadreze în timpul alocat, caz în care ne oprim după timpul alocat și afișăm soluția găsită pînă atunci în speranța că este cea optimă

Ideea de bază este să împărțim programul în segmente de calcul ce se execută iterativ, într-o buclă while. În acea buclă while vom testa timpul, ieșind din buclă dacă timpul rămas este mai mic de o milisecundă.

**Exemplul unu**

Iată un exemplu de program care încearcă să calculeze suma numerelor de la 1 la un miliard modulo 1234 avînd o secundă la dispoziție. El afișează numărul de iterații și suma calculată pînă la momentul respectiv:

#include <stdio.h>

#include <sys/time.h> // pentru structura timeval

#define MAXTIME 1000000 // timpul de executie maxim permis

long long start; // momentul de timp cind a pornit programul (in microsecunde)

// returneaza timpul in microsecunde (o milionime de secunda)

inline long long getTime() {

struct timeval tv;

gettimeofday(&tv, NULL);

return tv.tv\_sec \* 1000000LL + tv.tv\_usec;

}

int main() {

int i, suma;

start = getTime(); // timpul de inceput al programului, prima instructiune din main()

// ... urmeaza programul propriu-zis

// ... trebuie sa despartim calculul in ceva iterativ

// ... in bucla principala vom testa timpul

// ... iata un exemplu de program

suma = i = 0;

while ( i < 1000000000 && (getTime() - start) < (MAXTIME - 1000) ) {

suma = (suma + i) % 1234;

i++;

}

// ... am terminat codul mai devreme cu 1000 microsecunde (o milisecunda)

// ... pentru a nu risca sa depasim timpul

// ... deoarece vom mai face afisari, la final

printf( "Am apucat sa fac %d iteratii\n", i );

printf( "Suma pina la %d este %d\n", i, suma );

return 0;

}

Iată un exemplu de execuție pe un calculator cu procesor core 2 duo la 2GHz:

Am apucat sa fac 4309481 iteratii

Suma pina la 4309481 este 428

Deoarece împărțirile durează mult mai mult decît adunările am putea trage concluzia că pe acest calculator putem face circa patru milioane de împărțiri pe secundă. În realitate să nu uităm că și apelul funcției de obținere a timpului durează.

**Exemplul doi**

Deoarece testarea timpului ocupă și ea timp este bine să nu o facem foarte des. Dacă segmentul de calcul este extrem de scurt, cum ar fi în exemplul trecut o împărțire și o adunare, vom petrece timp foarte mult calculînd timpul. Dacă segmentul de calcul este foarte lung riscăm să depășim timpul în unul dintre segmente.

Cum putem modifica exemplul anterior pentru a lungim segmentul de calcul? Vom calcula împărțirile în grupe de cîte 1000. Deoarece împărțirile sînt totuși foarte rapide un astfel de segment de calcul va fi în continuare foarte rapid. Iată noua variantă:

#include <stdio.h>

#include <sys/time.h> // pentru structura timeval

#define MAXTIME 1000000 // timpul de executie maxim permis

#define BATCH 1000

long long start; // momentul de timp cind a pornit programul (in microsecunde)

// returneaza timpul in microsecunde (o milionime de secunda)

inline long long getTime() {

struct timeval tv;

gettimeofday(&tv, NULL);

return tv.tv\_sec \* 1000000LL + tv.tv\_usec;

}

int main() {

int i, j, suma;

start = getTime(); // timpul de inceput al programului, prima instructiune din main()

// ... urmeaza programul propriu-zis

// ... trebuie sa despartim calculul in ceva iterativ

// ... in bucla principala vom testa timpul

// ... iata un exemplu de program

i = suma = 0;

while ( i < 1000000000 && (getTime() - start) < (MAXTIME - 1000) ) {

for ( j = i; j < i + BATCH; j++ )

suma = (suma + j) % 1234;

i += BATCH;

}

// ... am terminat codul mai devreme cu 1000 microsecunde (o milisecunda)

// ... pentru a nu risca sa depasim timpul

// ... deoarece vom mai face afisari, la final

printf( "Am apucat sa fac %d iteratii\n", i );

printf( "Suma pina la %d este %d\n", i, suma );

return 0;

}

Pe același calculator vom obține rezultatul:

Am apucat sa fac 152449000 iteratii

Suma pina la 152449000 este 870

Iată că de la patru milioane de împărțiri pe secundă am sărit la 152 de milioane, ceea ce sună mult mai plauzibil.

Dacă modificăm programul să execute bucla în segmente de zece mii în loc de o mie (constanta BATCH obținem rezultatul:

Am apucat sa fac 157300000 iteratii

Suma pina la 157300000 este 622

Precum vedeți nu avem o variație prea mare: de la 152 de milioane am sărit la 157 de milioane. Putem spune cu certitudine că numărul de împărțiri pe secundă pe care le poate executa acest procesor este de circa 160 de milioane.

Concluzie

Am învățat să măsurăm timpul de execuție al unui program și să executăm un program vreme de o perioadă de timp fixată. Trebuie însă să reținem o lecție importantă, din fizică, care ne afectează în cazul de față:

**Efectul observatorului**

*Actul observării unui fenomen va schimba acel fenomen.*

Cu alte cuvinte, măsurarea unui sistem fizic nu se poate face fără a perturba acel sistem. Acesta este în multe cazuri rezultatul instrumentelor, care, din necesitate, alterează starea a ceea ce măsoară ele într-un fel sau altul. Un exemplu tipic este măsurarea presiunii unui cauciuc la mașină, care este greu de făcut fără a lăsa niște aer să scape, ceea ce schimbă presiunea originală. Spunem că într-un sistem fizic ceea ce măsurăm este, în realitate, un sistem format din sistemul original în care includem instrumentele folosite.

Efectul observatorului poate fi redus pînă la nesemnificativ folosind instrumente mai bune sau tehnici de observare mai bune. În cazul nostru am redus efectul măsurării folosind segmente mai mari de calcul.

**Reguli de programare în limbajul C**

Această secțiune este un *schelet*. Mai multe detalii în curînd.

* Variabilele simple **nu** se inițializează la declarare. Ele se inițializează cît mai aproape de secțiunea de program care le folosește. Cu alte cuvinte orice variabilă se inițializează cît mai jos posibil. De ce? Pentru citibilitatea codului. Imaginați-vă că la linia 300 vedem o instrucțiune a = a + y;. Ne punem întrebarea cu ce valoare a fost inițializată a. Imaginați-vă că trebuie să mergem cîteva pagini în sus pentru a constata că variabila a fost inițializată la declarare int a = 1;
* Vectorii încep de la 0, nu de la 1. Este o regulă importantă, nu o preferință personală. Avantaje: cînd folosim modulo, vectori caracteristici, etc.
* Vectorii au dimensiunile specificate de cerinţă. Ei nu se declară int v[10010]. Aceasta se cheamă *programare aproximativă* şi nu este o practică acceptată în unul din locurile de top din ţară. Dacă la matematică vi s-ar cere să rezolvaţi ecuaţia *2****·****x = 5* şi aţi răspunde că *x* este un număr între doi şi trei nu cred că aţi putea fi consideraţi cei mai mari matematicieni ai ţării. Programarea aproximativă reprezintă lenea minţii. *Să punem cîteva elemente în plus, în caz că depăşesc vectorul*. Cine depăşeşte vectorul? Zîna memoriei? Nu, voi. Soluţia corectă este să vă corectaţi gîndirea şi să vă ascuţiţi mintea pentru ca algoritmul vostru să **nu** depăşească vectorul. Altfel veţi rămîne incapabili de a scrie programe în lumea reală unde aproximarea nu e acceptabilă.
* Reguli de indentare, exemple:
  + Indentarea se face la două sau la patru spații. Două sînt suficiente pentru citibilitate. Atenție: setați code::blocks să nu folosească caractere TAB.
  + Acolada deschisă se pune pe același rînd cu instrucțiunea precedentă.
  + Acolada închisă se pune pe rîndul următor aliniată sub instrucțiunea de care aparține.
  + Dacă după **else** urmează un **if** atunci **if**-ul se pune imediat după **else** pe aceeași linie, iar corpul de instrucțiuni aparținînd lui **else** se indentează cu două spații, nu cu patru. Astfel, în cazul unor instrucțiuni de genul **if ... else if ... else if ...** se evită migrarea codului către dreapta prin indentări inutile.
* Fără break/continue/return/goto în interiorul buclelor. Ele distrug programarea structurată și, implicit, zeci de ani de experiență ai omenirii.
* Folosim **for** pentru bucle cu număr cunoscut de pași, **while** în celelalte cazuri. Aceasta este o regulă de programare ordonată, care face codul mai citibil.
* Nu avem voie să modificăm variabila de buclă **for**. Dacă este un ciclu cu număr cunoscut de pași nu avem de ce să modificăm variabila.
* Cînd eliminăm un element din vector scădem lungimea lui (n--). În felul acesta n continuă să arate numărul de elemente din vector.
* Nu aveți voie să aveți warnings, vă pot fi fatale: declaraţi int main() si folosiţi return 0 la final. Eliminați orice alte warnings deoarece majoritatea sînt o problemă reală. Unele concursuri compilează sursa cu opriri la warnings (adică tratează warning-urile ca erori). Unele warnings se referă la variabile neinițializate sau la număr diferit de "%d" versus variabile citite, in fscanf, deci sînt importante. Este **obligatoriu** să setaţi Code::Blocks să vă afişeze toate warnings. Pentru aceasta, după crearea proiectului, navigaţi în meniul *Project*, submeniul *Options*. În acea fereastră setaţi opţiunile *-Wall* şi *-O2*.
* Comentați codul cînd e ceva complicat, de exemplu o formulă. Scrieți ideea algoritmului, dacă nu e ceva trivial. Nu trebuie să umpleți de comentarii, dar minimal. Mă ajutați mult în corectarea temei.
* Denumiți variabilele mai clar. Nu lungi, dar să aibă semnificație: prod, sum, nrcf, etc.
* Nu abuzați stegulețele. Exemplu unde nu sînt necesare break sau stegulețe: căutarea unui element e în vectorul v:
* i = 0;
* while ( (i < n) && (v[i] != e) )

i++;

* Folosiți corect fișiere, le deschideți cu **fopen**, iar la final obligatoriu le închideți cu **fclose**. În nici un caz nu folosiți **freopen**.
* Nu folosiți funcții gen sort și qsort. Aveți nevoie de dispensă specială ☺

Recapitulare reguli de programare

Găsiți aici capitolul complet de reguli de programare: [reguli de programare în limbajul C](http://algopedia.ro/wiki/index.php/Reguli_ale_cercului_de_informatic%C4%83#Reguli_de_programare_.C3.AEn_limbajul_C)

Iată ce spune el:

* Variabilele simple **nu** se inițializează la declarare. Ele se inițializează cît mai aproape de secțiunea de program care le folosește. Cu alte cuvinte orice variabilă se inițializează cît mai jos posibil. De ce? Pentru citibilitatea codului. Imaginați-vă că la linia 300 vedem o instrucțiune a = a + y;. Ne punem întrebarea cu ce valoare a fost inițializată a. Imaginați-vă că trebuie să mergem cîteva pagini în sus pentru a constata că variabila a fost inițializată la declarare int a = 1;
* Vectorii încep de la 0, nu de la 1. Este o regulă importantă, nu o preferință personală. Avantaje: cînd folosim modulo, vectori caracteristici, etc.
* Reguli de indentare, exemple:
  + Indentarea se face la două sau la patru spații. Două sînt suficiente pentru citibilitate. Atenție: setați code::blocks să nu folosească caractere TAB.
  + Acolada deschisă se pune pe același rînd cu instrucțiunea precedentă.
  + Acolada închisă se pune pe rîndul următor aliniată sub instrucțiunea de care aparține.
  + Dacă după **else** urmează un **if** atunci **if**-ul se pune imediat după **else** pe aceeași linie, iar corpul de instrucțiuni aparținînd lui **else** se indentează cu două spații, nu cu patru. Astfel, în cazul unor instrucțiuni de genul **if ... else if ... else if ...** se evită migrarea codului către dreapta prin indentări inutile.
* Fără break/continue/return/goto în interiorul buclelor. Ele distrug programarea structurată și, implicit, zeci de ani de experiență ai omenirii.
* Folosim **for** pentru bucle cu număr cunoscut de pași, **while** în celelalte cazuri. Aceasta este o regulă de programare ordonată, care face codul mai citibil.
* Nu avem voie să modificăm variabila de buclă **for**. Dacă este un ciclu cu număr cunoscut de pași nu avem de ce să modificăm variabila.
* Cînd eliminăm un element din vector scădem lungimea lui (n--). În felul acesta n continuă să arate numărul de elemente din vector.
* Nu aveți voie să aveți warnings, vă pot fi fatale: declaraţi int main() si folosiţi return 0 la final. Eliminați orice alte warnings deoarece majoritatea sînt o problemă reală. Unele concursuri compilează sursa cu opriri la warnings (adică tratează warning-urile ca erori). Unele warnings se referă la variabile neinițializate sau la număr diferit de "%d" versus variabile citite, in fscanf, deci sînt importante. Puteți seta Code::Blocks să afișeze toate warnings. Este **obligatoriu** să faceți acest lucru.
* Comentați codul cînd e ceva complicat, de exemplu o formulă. Scrieți ideea algoritmului, dacă nu e ceva trivial. Nu trebuie să umpleți de comentarii, dar minimal. Mă ajutați mult în corectarea temei.
* Denumiți variabilele mai clar. Nu lungi, dar să aibă semnificație: prod, sum, nrcf, etc.
* Nu abuzați stegulețele. Exemplu unde nu sînt necesare break sau stegulețe: căutarea unui element e în vectorul v:
* i = 0;
* while ( (i < n) && (v[i] != e) )

i++;

* Nu folosiți funcții gen sort și qsort. Aveți nevoie de dispensă specială ☺

Generalități

Să ne aducem aminte următoarele lucruri, căci vom avea nevoie de ele la tema:

* Declararea vectorilor cu inițializare.
* Dimensiunile diverselor tipuri: în code::blocks **int** are 4 octeți, **long long** are 8 octeți.
* Estimarea numărului care încape pe un număr dat de biți folosind formula 210 ≈ 103

Sfaturi pentru olimpiadă

Iată cîteva sugestii pentru olimpiadă.

Ce să faceți / ce să nu faceți

**Ce să faceți**

* **Odihnă**: odihniți-vă cu o zi înainte. Relaxați-vă cu activitatea favorită. Mergeți la un film, jucați jocul vostru preferat, etc. Încercați să nu vă gîndiți la concurs.
* **Ceas**: aveți un ceas la voi. Nu ceasul calculatorului, al telefonului sau al smartphone-ului. Fiți conștienți de trecerea timpului, nu vă treziți din visare cînd mai este un sfert de oră. Pentru aceasta un ceas așezat pe banca de lucru este ideal. Nu uitați: toate celelalte ceasuri pot fi incorecte, sau să moară.
* **Țineți socoteala timpului**: notați ora începerii concursului pe foaie. Verificați la final că ați avut tot timpul promis. Dacă aveți probleme cu calculatorul și cereți ajutorul supraveghetorului notați timpul cînd ați fost întrerupți și apoi timpul cînd reveniți la un calculator funcțional. Aveți dreptul la o prelungire a timpului egală cu timpul pierdut.
* **Cereți-vă drepturile**: dacă vă opresc mai devreme decît e cazul, sau dacă ați pierdut zece minute din cauza unui calculator care nu a funcționat și nu vi se dau, cereți respectuos să vi se extindă timpul cu acele minute. Supraveghetorii pot uneori să uite, olimpiada este stresantă și pentru ei. Dacă nu înțelegeți textul unei probleme, după ce l-ați citit cu mare atenție, puneți întrebări pentru clarificare (formulate astfel încît răspunsul să fie 'DA' sau 'NU').
* **Low hanging fruit**: rezolvați problema mai ușoară prima. Citiți toate problemele la început și hotărîți-vă care este mai ușoară. Cu ea începeți. Puteți inclusiv să rezolvați subpunctele mai ușoare primele. De exemplu puteți să rezolvați problema 1 punctul a), apoi problema 2 punctul b).
* **Moral ridicat**: intrați în sală încrezători în voi. Sînteți printre cei mai buni din țară, aveți puterea să vă clasați printre primii din țară. Propuneți-vă să luați punctaj maxim și reamintiți-vă că puteți acest lucru.
* **Probleme grele**: atunci cînd constatați că problemele sînt grele nu vă speriați. Bucurați-vă! Este cea mai bună situație pentru voi. Aveți numai de cîștigat: fie știți să faceți problema, caz în care este perfect, căci problema fiind grea v-ați asigurat că veți fi mai sus ca ceilalți. Fie nu știți să faceți problema și atunci care este șansa ca cei mai slabi ca voi să știe să o facă? Temeți-vă de problemele ușoare, pe care știu toți să le facă. Voi le veți gîndi, ceilalți le vor picta la perfecție.
* **Atenție**: fiți foarte atenți la detalii: la numele fișierelor de intrare/ieșire; la numele sub care trebuie să salvați programul C; la setările de proiect CodeBlocks de care am vorbit: -Wall și -O2; la locul unde ați salvat proiectul, și numele proiectului în caz că aveți nevoie să le copiați în altă parte.

**Ce să nu faceți**

* **Nu îngrășați porcul în ajun**: nu faceți probleme în ultima clipă . Vă veți extenua inutil. În urma stresului s-ar putea să nu reușiți să le faceți, ceea ce vă va demoraliza. Aveți nevoie de moral ridicat la concurs.
* **Nu vă panicați**: panica ucide creierul. Panica vă face să gîndiți mai încet și să doriți în subconștient să ieșiți din sală. Controlați-o. Nu uitați că o problemă grea, pe care nu știți să o faceți, vă avantajează, pentru că probabil nici ceilalți nu știu, pe cînd voi aveți o șansă să luați puncte pe ea.
* **Nu-i băgați în seamă pe cei ce se dau mari**: unii elevi spun cu voce suficient de tare să fie auziți, în timpul concursului, lucruri de genul 'aaah, am făcut-o și pe asta, super' sau 'am rupt, sigur iau suta', etc. Este modul lor inconștient de a vă demoraliza. De obicei ei se clasează slab. Ignorați-i. De asemenea veți vedea că unii dintre concurenți părăsesc sala mai devreme cu o oră cu un aer fericit, eventual spunînd 'sigur mă calific' sau 'am făcut tot, yesss'. Ignorați-i. Și ei se clasează slab. Nu vă demoralizați indiferent cîți elevi ies din sală. Lupta voastră nu este cu ei, ci cu problemele și punctele.
* **Nu ieșiți mai devreme din sală**: dacă nu ați terminat problemele continuați să vă gîndiți la o soluție. Dacă ați terminat problemele și vă plictisiți creați mai multe teste pe care să le testați. Nu uitați: nimănui nu-i pasă cît de repede ați ieșit. Nu luați puncte suplimentare pentru asta.
* **Nu modificați în ultima clipă un program care funcționează**: riscați să nu aveți timp să-l testați și să luați zero puncte la acea problemă.
* **Nu vă îngrijorați de părerea noastră**: noi, instructorii, știm bine care este valoarea voastră. Nu avem nevoie de un rezultat la olimpiadă să aflăm. Nu o să ne vedeți vreodată supărați pe voi pentru că nu ați luat punctaj bun. Dimpotrivă, orice veți face vom fi mîndri de voi pentru cît de departe ați ajuns într-un timp atît de scurt. Grija voastră trebuie să fie rezolvarea problemelor, nu noi. Noi sîntem de partea voastră. Chiar dacă la cerc sau la clasă, cînd sîntem între noi, vă certăm sau ne mai supărăm pe voi, în momentul cînd ați ieșit de la cerc să vă "luptați" cu olimpiada pentru noi, sîntem de aceeași parte a baricadei.

Diferenţa dintre olimpiadă şi viaţa reală

Nu uitaţi că olimpiada este un concurs de verificare a cunoştinţelor, nimic mai mult, nimic mai puţin. Ea nu este evaluarea supremă a unui elev, la fel cum nu este nici ceva de ignorat cu dispreţ. Este cel mai mare concurs la gimnaziu și liceu, drept pentru care îl vom trata cu respect. Dar nu ne vom spînzura de grindă, considerînd că viaţa s-a sfîrşit, dacă nu obţinem rezultatul dorit. **Olimpiada nu ne defineşte ca oameni**.

Acestea fiind spuse, dacă mergem la olimpiadă, ne dorim un scor cît mai bun. De aceea consider utilă următoarea analogie între arte marţiale si informatică:

* **În viaţa reală**: în arte marţiale învăţăm pe viaţă. Procedee noi, ce trebuie aplicate cît mai corect, perfect dacă se poate. Filozofie, mod de comportament, respect faţă de artă şi luptă. Atunci cînd mişcarea nu este perfectă, instructorul vă va corecta. Bătaia este interzisă cu excepţia cazului cînd sînteţi în legitimă apărare. Este imoral şi foarte urît să cîştigaţi avantaje în viaţa reală bătîndu-i pe cei mai slabi. Artele marţiale au ca scop îmbunătăţirea personală.
* **La olimpiadă**: olimpiada este ca legitima apărare. Ca şi cînd sînteţi cu fratele mai mic după voi şi vă atacă golanii. În acel moment nu vă verifică nimeni cît de frumos aţi executat procedeul pentru a scăpa teferi. Tot ce contează este să învingeţi. Pentru aceasta vă veţi folosi de orice! Dacă găsiţi un bolovan, îl aruncaţi în adversar. O chiuvetă spartă, bună şi aceea! La fel şi la olimpiadă, tot ce contează sînt punctele. Nu cît de frumoasă este metoda folosită, nici cît de frumos aţi scris codul. Dacă puteţi lua puncte afişînd mereu "0", o faceţi. Această practică nu ar fi acceptată nici la cercul nostru şi nici în viaţa reală. Din păcate, însă, la olimpiadă nu se se punctează stilul şi perfecţionismul, ci numai testele trecute.

Lucru individual: subiect BAC 2010...2020

Invatare:

Cormen(CLR)==MasachusetsIT

<https://www.geeksforgeeks.org/>

<https://codeforces.com/problemset>