Proiectarea structurata top-down a programelor complexe

Aplicatie

Sa se scrie un program care citeste un fisier continand un text format din cuvinte si determina si afiseaza grupe de cuvinte, grupate in functie de un criteriu care se selecteaza.

Criteriile de grupare pot fi: apartin unei grupe

- *cuvintele care au acelasi numar de litere*
- *cuvintele care au aceleasi numere de aparitii de vocale*
- *cuvintele care sunt anagrame.*

Precizari:

1. *La cuvinte nu se face deosebire intre litera mica si litera mare.*
2. *Doua cuvinte se numesc anagrame daca ele sunt formate din exact aceleasi caractere in alta ordine. Exemplu de grupa de anagrame: "mar" "ram" "arm"*
3. *Doua cuvinte au aceleasi numere de aparitii de vocale, daca fiecare din vocale apare de exat acelasi numar de ori in ambele cuvinte. Exemplu: "aflu" si "sufla"*
4. *Cuvintele pot contine doar caractere litere si pot fi separate prin spatii sau linie noua. Intre doua cuvinte pot exista oricate spatii.*

Programul va primi 2 parametrii in linia de comanda: numele fisierului de prelucrat si modul de prelucrare (specificat prin "anagrame", "nrvoc" sau "lungimi"). Se vor afisa grupele de cuvinte conform criteriului specificat. Grupele de cuvinte pot fi afisate utilizand doar caractere litere mici.

De exemplu, daca programul se numeste grupe, pentru a determina grupele de cuvinte din fisierul fis.txt care sunt anagrame se va tasta in linia de comanda

> *grupe fis.txt anagrame*

Pentru a afisa grupele de cuvinte care au aceeasi lungime se da comanda

> *grupe fis.txt lungimi*

De la specificarea problemei de rezolvat la program

Intre specificarea problemei si scrierea programului de rezolvare nu e o trecere directa, ci exista mai multe etape.

In primul rand, e necesara o etapa de analiza si abstractizare a problemei, de identificare a subproblemelor in care se poate descompune problema initiala,

subprobleme care, la randul lor, se pot descompune in alte subprobleme. Pentru fiecare subproblema se identifica obiectelor din domeniul problemei implicate in rezolvare si actiunile de transformare corespunzatoare acestora.

Urmeaza apoi etapele gasirii unor metode de rezolvare pentru fiecare subproblema, elaborarii algoritmilor de rezolvare si in final codificarea algoritmilor si reprezentarii obiectelor din universul problemei ca si tipuri de date (abstracte) folosind facilitatile oferite de limbajul de programare.

Un program C este format din ansamblul fisierelor de program care grupeaza toate functiile necesare rezolvarii problemei. Un fisier de program e numit modul program. Gruparea functiilor unui program in module se face in general grupand intr-un modul functiile referitoare la rezolvarea unei subprobleme sau functiile care realizeaza transformarile caracteristice unui tip de obiect reprezentat ca si tip de date abstract.

Nu exista o unica solutie pentru problema cum se grupeaza functiile unui program pe module, sau cum se modularizeaza programul. Un program structurat este constituit din unitati functionale bine definite – module, ierarhizate conform naturii intrinseci a problemei.

Analiza problemei

Identificarea datelor de intrare: datele de intrare sunt numele fisierului text din care se citeste, si criteriul de grupare. Posibile date de intrare eronate: daca numele fisierului e un fisier inexistent sau acel fisier nu poate fi deschis pentru citire, sau daca criteriul de grupare introdus nu e unul din cele 3 permise.

Specificarea functionala: in primul rand, trebuie identificate toate cuvintele din fisier. Deoarece se mentioneaza faptul ca nu se face deosebire intre litere mici si litere mari, se poate face o conversie a tuturor cuvintelor in scriere cu litere mici. Pentru fiecare cuvant, trebuie gasita grupa de care apartine. Aici intervine criteriul de grupare dat ca si parametru. Pentru fiecare cuvant, se cauta daca poate fi inclus, conform criteriului de grupare, in una din grupele deja existente. Daca nu poate fi adaugat la o grupa existenta, se infiinteaza o noua grupa cu acest cuvant.

Datele de iesire: grupele de cuvinte, se afiseaza.

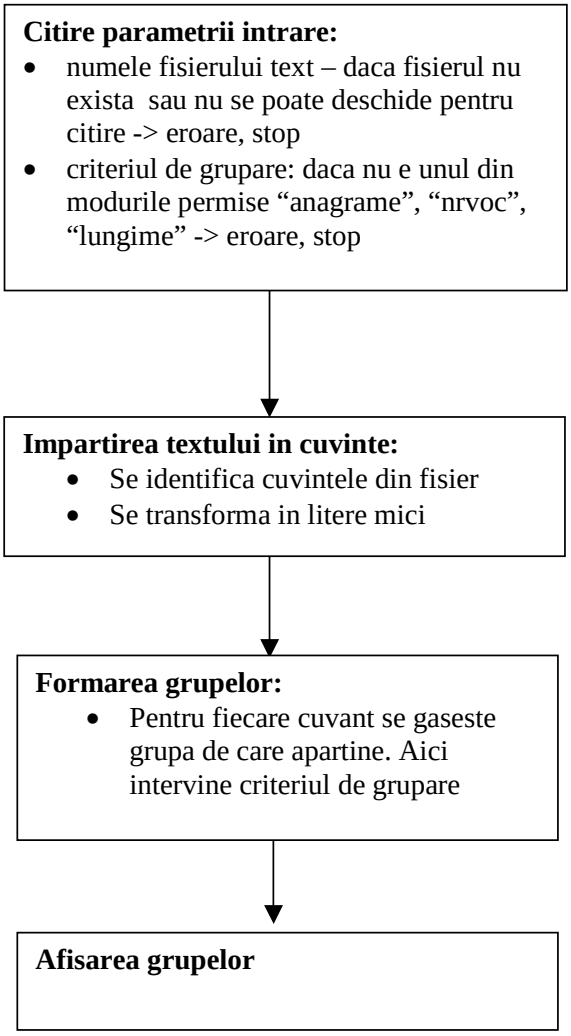


Figure 1

Proiectarea solutiei

Proiectarea solutiei incepe cu identificarea subproblemelor si stabilirea modulelor programului.

Modulul principal (**`grupe`**) contine functia main, care realizeaza interpretarea parametrilor si selecteaza un anumit criteriu de grupare. Criteriul de grupare va fi realizat sub forma unei functii cu 2 parametrii (2 cuvinte) si care returneaza daca cele 2 cuvinte trebuie sa apartina aceleiasi grupe sau nu.

Pentru descrierea criteriului de grupare, se defineste un tip functie

```
typedef int (*functie_test)( char * cuv1, char* cuv2);
```

Criteriul de grupare trebuie sa fie cunoscut si de catre gestionarul listei de grupe, deci va fi pus intr-un fisier header separat, **`tip_fct`**.

Fiecare criteriu de grupare poate fi implementat ca si un modul separat. Modulul **`anagrame`** si modulul **`nrvoc`**, fiecare dintre acestea exporta cate o functie de tipul functiei criteriu de grupare `functie_test`,

```
int anagrame  (char * cuv1, char * cuv2);
int nrvoc(char * cuv1, char * cuv2);
```

Criteriul de comparare a doua cuvinte pe baza lungimii lor e mai simplu, poate fi implementat printr-o functie simpla in cadrul modulului principal.

Un modul separat **`prel_fis`** realizeaza prelucrarea fisierului,in sensul separarii cuvintelor din text. Functia exportata de acest modul este

```
void determina_grupe(FILE *, functie_test);
```

si este apelata de catre programul principal. Functia are 2 parametri, fisierul deschis pentru citire si functia criteriu de grupare. Modulul de prelucrare a fisierului utilizeaza facilitatile implementate de alt modul, gestionarul de grupe.

Gestiunea listei de grupe este facuta intr-un modul separat, **`lista_g`**. Cele 2 operatii realizate de acest modul sunt:

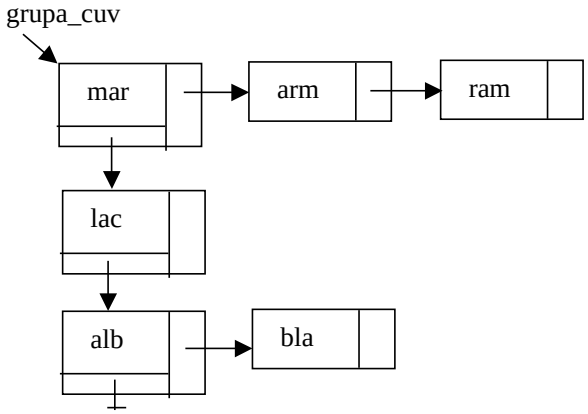
```
void adauga_grupat(char * cuvant, functie_test criteriu);
```

Aceasta functie adauga cuvantuldat ca si parametru in functie de criteriul dat in una din grupe. Daca cuvantul poate fi adaugat la o grupa existenta, il adauga, daca nu creaza o noua grupa cu el.

```
void afiseaza_grupe();
```

Aceasta functie tipareste toate grupele existente.

Modul de implementare a listei de grupe este un *detaliu intern* al modulului `lista_g`. Gestionarea grupelor se poate face utilizand diverse structuri de date, atat dinamice cat si statice. Se va opta pentru o structura de date dinamica, de tip multilista.



Fiecare modul este realizat sub forma unui fisier independent, interfata cu celelalte module e asigurata prinfisierele antet asociate.

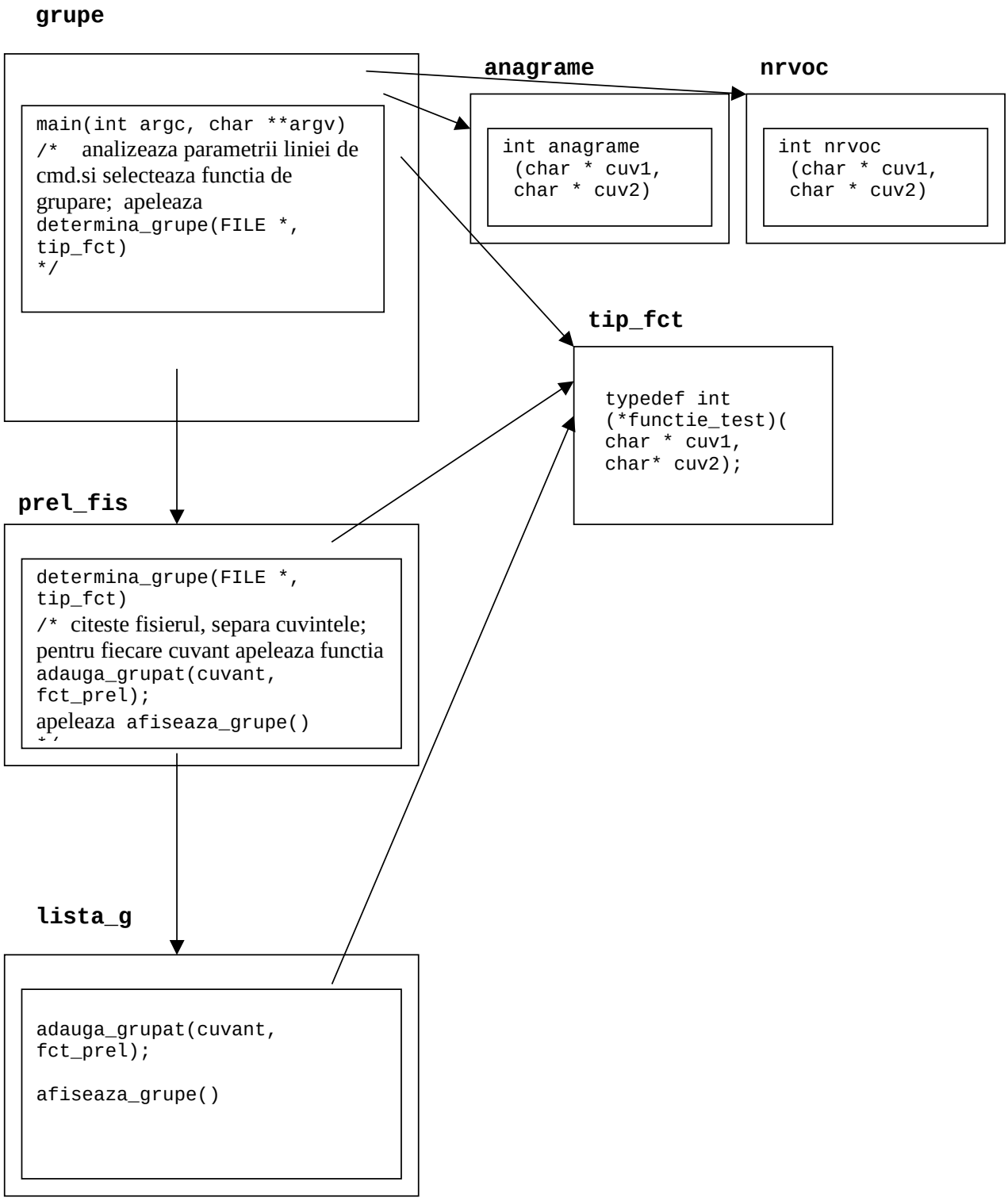

grupe
main(int argc, char **argv)
/* analizeaza parametrii liniei de cmd.si selecteaza functia de grupare; apeleaza determina_grupe(FILE *, tip_fct)
*/
anagrame
int anagrame (char * cuv1, char * cuv2)
nrvoc
int nrvoc (char * cuv1, char * cuv2)
tip_fct
typedef int (*functie_test)(char * cuv1, char* cuv2);
prel_fis
determina_grupe(FILE *, tip_fct)
/* citeste fisierul, separa cuvintele; pentru fiecare cuvant apeleaza functia adauga_grupat(cuvant, fct_prel);
apeleaza afiseaza_grupe()
lista_g
adauga_grupat(cuvant, fct_prel);
afiseaza_grupe()

Figure 2

Codificarea modulelor

```
/* grupe.c   */

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

#include "tip_fct.h"
#include "prel_fis.h"
#include "anagram.h"
#include "nrvoc.h"

int lungimi(char * cuv1, char* cuv2) {
      return strlen(cuv1)==strlen(cuv2);
}

void main(int argc, char * argv[]) {
      char *nume_fis, *mod;
      FILE * fp;

      if (argc!=3)
            printf("Utilizarea programului: grupe fisier mod");
      else {
            nume_fis=argv[1];
            mod=argv[2];
            if ((fp=fopen(nume_fis, "r"))==NULL) {
                  printf("Nu pot deschide fisierul %s \n",
nume_fis);
                  exit(1);
                  }
            if (strcmp(mod, "anagrame")==0)
                  determina_grupe(fp, anagrame);
            else if (strcmp(mod,"lungimi")==0)
                  determina_grupe(fp, lungimi);
            else if (strcmp(mod,"nrvoc")==0)
                  determina_grupe(fp, nrvoc);
            else {
                  printf("Mod de prelucrare necunoscut \n");
                  exit(1);
                  }
            }
      }
```

```
/* tip_fct.h */

typedef int (*functie_test)(char * cuv1, char* cuv2);
```

```
/* prel_fis.h */

#include <stdio.h>
#include "tip_fct.h"

void determina_grupe( FILE * fp, functie_test functie);
```

```
/* prel_fis.c */

#include <stdio.h>

#include "tip_fct.h"
#include "lista_g.h"

#define ESPATIU(c) (c==' ' || c=='\n')
#define INCUVANT(c) ((c>='a' && c<='z')||(c>='A' && c<='Z'))
#define LITMICA(c) (c>='a' && c<='z') ? (c): (c+'a'-'A');

void determina_grupe( FILE * fp, functie_test functie) {
int c, i;
char buf[100];
            c=getc(fp);
            while (c!=EOF) {
                  while (ESPATIU(c))
                        c=getc(fp);
                  if (c==EOF) break;
                  i=0;
                  while (INCUVANT(c)) {
                        buf[i++]=LITMICA(c);
                        c=getc(fp);
                  }
                  buf[i]='\0';
                  // printf(" Cuvantul %s \n", buf);
                  adauga_grupat(buf, functie);
                  }
             afiseaza_grupe();
      }
```

```
/* lista_g.h */

#include "tip_fct.h"

void adauga_grupat(char * cuv, functie_test functie) ;
void afiseaza_grupe(void);
```

```
/* lista_g.c */

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

#include "lista_g.h"

typedef struct cv{
      char * cuv;
      struct cv * urm;
      } cuvant;

typedef struct gc{
      char * cuv;
      cuvant * grupa;
      struct gc *urm;
      } grupa_cuv;
```

```
static grupa_cuv * lista=NULL;

void adauga_grupat(char * cuv, functie_test functie) {
      grupa_cuv * l, *noug;
      cuvant *nouc;
      l=lista;
      while ((l!=NULL)&&(!functie(l->cuv, cuv)))
            l=l->urm;
      if (l==NULL)      {
            noug=(grupa_cuv *)malloc(sizeof(grupa_cuv));
            noug->cuv=(char *)malloc(strlen(cuv)+1);
            strcpy(noug->cuv, cuv);
            noug->urm=lista;
            noug->grupa=NULL;
            lista=noug;
      }
      else {
            nouc=(cuvant *)malloc(sizeof(cuvant));
            nouc->cuv=(char *)malloc(strlen(cuv)+1);
            strcpy(nouc->cuv, cuv);
            nouc->urm=l->grupa;
            l->grupa=nouc;
      }
}


void afiseaza_grupe(void) {
grupa_cuv * l=lista;
cuvant * cv;
printf("\n GRUPELE: \n");
while (l!=NULL) {
      printf("%s ", l->cuv);
      cv=l->grupa;
      while (cv!=NULL) {
            printf("%s ", cv->cuv);
            cv=cv->urm;
            }
      printf("\n");
      l=l->urm;
      }
}
```

```
/*anagram.h */

int anagrame(char * cuv1, char* cuv2);
```

```
/* anagram.c*/

#include <string.h>

static void contoare(char * cuv, int c[]) {
int i;
for (i=0; i<27; i++)
      c[i]=0;
for (i=0; i<strlen(cuv); i++)
      c[cuv[i]-'a']++;
}
```

```
static int egal (int c1[], int c2[]) {
int i;
for (i=0; i<27; i++)
      if( c1[i]!=c2[i]) return 0;
return 1;
}

int anagrame(char * cuv1, char* cuv2) {
      int c1[27];
      int c2[27];
      contoare(cuv1, c1);
      contoare(cuv2, c2);
      return egal(c1, c2);
}
```

/* nrvoc.h */

```
int nrvoc(char * cuv1, char* cuv2);
```

```
/* nrvoc.c */

#include <string.h>

#define NRLIT 5

static void contoare(char * cuv, int c[]) {
int i;
for (i=0; i<NRLIT; i++)
      c[i]=0;
for (i=0; i<strlen(cuv); i++)
      switch (c[i]) {
            case 'a': c[0]++; break;
            case 'e': c[1]++; break;
            case 'i': c[2]++; break;
            case 'o': c[3]++; break;
            case 'u': c[4]++; break;
            }
}

static int egal (int c1[], int c2[]) {
int i;
for (i=0; i<NRLIT; i++)
      if( c1[i]!=c2[i]) return 0;
return 1;
}

int nrvoc(char * cuv1, char* cuv2) {
      int c1[NRLIT];
      int c2[NRLIT];
      contoare(cuv1, c1);
      contoare(cuv2, c2);
      return egal(c1, c2);
}
```