Compilarea independenta

Functiile unui program nu sunt intotdeauna grupate in acelasi fisier. De exemplu, functiile bibliotecilor standard, care sunt utilizate in majoritatea programelor, sunt compilate separate si sunt incorporate in programul executabil la editarea de legaturi (linkeditare).

Fazele de prelucrare a fisierelor unui program C sunt: preprocesarea, compilarea si linkeditarea.

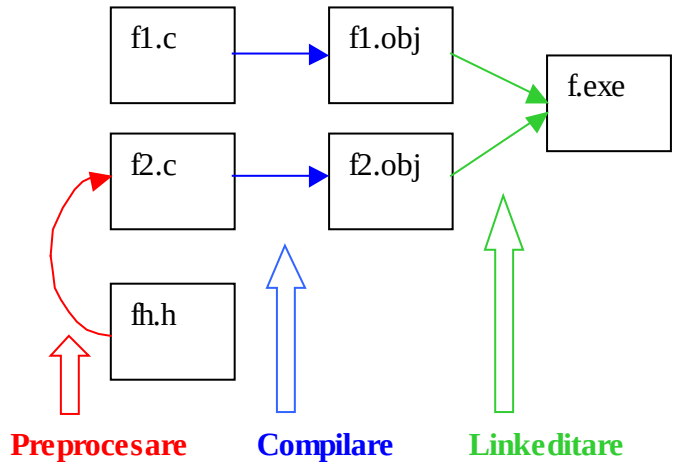

Figure 1

Diferitele parti ale unui program trebuie sa foloseasca consistent entitatile sale: un nume care nu e local unei functii trebuie sa se refere la aceeasi entitate (variabila, functie, tip) indiferent daca apare intr-un singur fisier sau mai multe. Problemele care trebuie rezolvate pentru a putea avea un program impartit in mai multe fisiere sursa sunt:

- cum se asigura faptul ca o variabila globala `x` definita intr-un fisier poate fi cunoscuta la compilare in alte fisiere ? cum se asigura faptul ca la linkeditare aceeasi zona unica de memorie este asociata acestei variabile ?
- cum se asigura faptul ca o functie definita intr-un fisier poate fi apelata in alte fisiere ?

Pe langa modulele program, care contin definitiile functiilor, exista module *antet(header)* asociate modulelor program. Aceste module antet au rol declarativ. Un modul antet asociat unui modul program contine toate informatiile necesare compilatorului pentru a verifica corectitudinea utilizarii functiilor sau altor entitati de program definite in modulul program asociat, atunci cand acestea sunt utilizate in alte module.

Daca un modul program `m1.c` contine functiile `f1, f2, …., fn`, functii apelate din alte module, cum sunt `m2.c`, modulul antet `m1.h` atasat lui `m1.c` trebuie sa contina declaratiile acestor functii. O declaratie a unei functii indica numarul si tipul parametrilor functiei, tipul rezultatului, fara a da definitia corpului functiei. Modulul antet ofera posibilitatea de a verifica la compilare faptul ca utilizarea acestor entitati din alte module se face corect.

Un exemplu deja cunoscut de compilare independenta e in cazul utilizarii functiilor de biblioteca. Acestea sunt grupate, in functie de tipurile de date prelucrate si de operatii, fiecare biblioteca avand un modul antet corespunzator:

Functii de biblioteca	**Antet**
Prelucrare de siruri de caractere	String.h
Intrari/iesiri standard	Stdio.h
Gestiunea dinamica a memoriei	Alloc.h

I
In mod asemanator, pot exista si module scrise de utilizatori care sa exporte functii sau date. De obicei, declaratiile acestor entitati exportate se grupeaza intr-un fisier antet asociat. Se obisnuieste ca numele fisierului antet asociat unui program modul "x" sa fie "x.h" iar numele modulului program"x.c".

Un fisier antet poate contine:

- Definitii de tipuri
- Declaratii de functii
- Declaratii de variabile
- Definitii de constante
- Macrodefinitii
- Directive de includere

Un fisier antet **nu** trebuie sa contina **definitii** de functii si definitii de date !
Se reaminteste faptul ca ***definitia*** si ***declaratia*** unei variabile externe sunt operatii diferite!
O definitie creaza variabila, ii rezerva spatiu de memorie.
O declaratie precizeaza doar natura variabilei, dar fara sa ii reserve spatiu de memorie.
O anumita variabila poate avea oricate declaratii, dar poate fi definita o singura data. Daca se include definitia unei variabile globale intr-un header, si acest header este inclus de mai multe ori in fisiere ale aceluiasi proiect, se va obtine o eroare de compilare din cauza variabilei definite de mai multe ori.

Variabile externe

Variabilele externe exista si isi pastreaza valorile de-a lungul executiei intregului program. Ele pot fi folosite pentru comunicarea intre functii, chiar daca functiile sunt compilate separate. Pentru identificatorii din clasa extern, trebuie sa existe o definitie in unul din fisiere, in celelalte fisiere apare doar declaratia acestora in care se foloseste specificatorul extern.

Exemplul urmator prezinta 2 functii (`calculeaza()` si `main()`) aflate in fisiere diferite si compilate separat, care folosesc in comun o variabila externa x. Memoria corespunzatoare lui x este alocata prin definitia din fisierul `progr.c`. Declaratia lui x din fisierul `calcule.c` precizeaza (prin modificatorul `extern`) ca memoria pentru x este alocata in alta parte.

```
/* fisier calcule.c */
```

```
extern int x;
void calculeaza(void) {
      printf(“%d ”, x+1);
}

/* fisier progr.c */

int x;
void main() {
x=7;
calculeaza();
}
```

In general, daca o functie se refera la un nume declarat extern, atunci trebuie sa existe o definitie a acelui identificator intr-unul din fisierele sau bibliotecile programului. Toate functiile unui program care se refera la acelasi nume extern indica aceeasi entitate. Entitatilor declarate cu extern nu li se mai aloca memoriela compilare, dar in faza de linkeditare sunt asociati definitiilor lor. Tipul specificat in declaratiile externe trebuie sa fie acelasi ca tipul din definitie. Intr-un program compus din mai multe fisiere,o definitie de date externe, fara specificatorul extern, are voie sa apara o singura data, intr-un singur fisier. Orice alt fisier care mai declara acel identificator trebuie sa foloseasca cuvantul cheie extern in declaratie. Omiterea cuvantului cheie extern din declaratiile ulterioare e eroare, va fi semnalata ca eroare la linkeditarea fisierelor impreuna – respectivul nume va apare redefinit de mai multe ori !

```
/* fisier globals.h */
int x;
…

/* fisier c1.c */
#include “globals.h”
…

/* fisier c2.c */
#include “globals.h”
…
```

Variabile si functii statice

Declaratia ca static a unei variabile externe sau functii, ii limiteaza durata de viata la fisierul curent. O asemenea variabila sau functie nu poate fi apelata din alt fisier sursa. O variabila sau o functie statica nu este vizibila din alte fisiere.

(Declaratia de clasa static se poate aplica si variabilelor interne (locale unei functii). O asemenea variabila va exista inpermanenta, in loc sa fie creata si sa dispara la fiecare apel al functiei. Acest tip de variabile sunt un mijloc de a stoca date pentru uzul exclusiv al unei anumite functii.)

In concluzie, la impartirea codului sursa a unui program intre mai multe fisiere, apar unele probleme legate de:

- Cum sa se scrie declaratiile pentru ca la compilare toate variabilele sa fie correct declarate
- Cum si unde sa se distribuie declaratiile, pentru ca partile sa fie imbinate correct de catre editorul de legaturi (linkeditare)

- Cum trebuie facute declaratiile astfel incat ele sa apara la linkeditare ca unice

Aplicatie. *O biblioteca de functii care implementeaza operatii cu polinoame.*
Operatiile cu polinoame sunt:
- adunarea a doua polinoame
- inmultirea a doua polinoame
- copierea unui polinom in alt polinom
- afisarea valorii unui polinom intr-un punct dat
- citirea unui polinom
- afisarea unui polinom

Se alege ca structura de date pentru reprezentarea unui polinom tabloul care memoreaza coeficientii polinomului. Fie tabloul coef, in acest fel, `coef[i]` reprezinta coeficientul termenului de grad i. Aceste tablouri sunt alocate static, polinoamele pot fi de gradul maxim `MAXGRAD`.

Intr-un fisier antet `polinom.h`, se defineste tipul structurat `polinom`, ca fiind tabloul coeficientilor si gradul polinomului, si se declara prototipurile functiilor care realizeaza operatiile cu polinoame.

```
/* polinom.h */

#define MAXGRAD 10

typedef struct{
      int grad;
    int coef[MAXGRAD];
} polinom;

polinom copie(polinom srs);
polinom  adunare(polinom, polinom);
polinom  inmultire(polinom, polinom);
float valoare(polinom, float);

polinom citire(void);
void afisare(polinom);
```

Dintre operatiile pe polinoame, implementarile lor vor fi grupate in doua fisiere separate: un fisier, `polinom.c`, va contine operatiile matematice de baza cu polinoame (adunare, inmultire, valoare), iar un al doilea fisier, `polinomio.c`, va contine operatiile de intrare-iesire (citire, afisare) pentru polinoame. Aceste functii de lucru cu polinoame vor putea fi utilizate din diferite programe principale. Fiecare fisier poate fi compilat in mod independent. Pentru a obtine programul executabil, toate fisierele trebuie linkeditate impreuna.

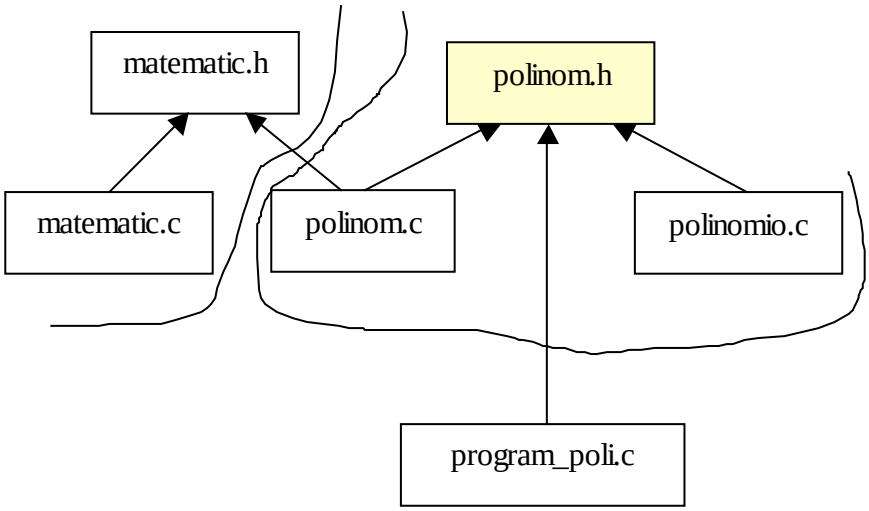

Figure 2. Compilarea independenta. Relatii de tip "include"

Functiile reprezentand operatiile matematice de baza asupra polinoamelor sunt definite in fisierul polinom.c. Pentru realizarea acestor functii, se definesc si unele functii ajutatoare. Acestea nu trebuie sa fie vizibile in afara fisierului de implementare polinom.c, deci vor fi declarate ca fiind functii de clasa `static`. De asemenea, unele operatii asupra polinoamelor utilizeaza si functii matematice care sunt definite intr-o alta biblioteca, matematic.c (de exemplu functia de ridicare la putere), si de aceea se include fisierul antet `matematica.h`.

```
/* polinom.c */

#include "polinom.h"
#include "matematic.h"
#include <stdio.h>

static polinom polinom_nul(){
 int g;
 polinom p;
 p.grad=0;
 for (g=0; g<MAXGRAD; g++)
        p.coef[g]=0;
 return p;
}

polinom adunare(polinom p1, polinom p2) {
 int g;
 polinom s=polinom_nul();
 if (p1.grad>p2.grad) s.grad=p1.grad;
 else s.grad=p2.grad;
 for (g=0; g<=s.grad; g++) {
       s.coef[g]=p1.coef[g]+p2.coef[g];
      }
 return s;
}

float valoare(polinom p, float x) {
 int g;
 float s=0;
 for (g=0; g<=p.grad; g++)
```

```
        s=s+p.coef[g]*putere(x, g);
 return s;
}

polinom copie(polinom srs) {
 int g;
 polinom dest=polinom_nul();
 dest.grad=srs.grad;
 for (g=0; g<=srs.grad; g++)
        dest.coef[g]=srs.coef[g];
 return dest;
}

static polinom inm_termen(int coef, int grad, polinom p) {
 int g;
 polinom r=polinom_nul();
 r.grad=p.grad+grad;
 for (g=0; g<=p.grad; g++) {
        r.coef[g+grad]=p.coef[g]*coef;
 }
 return r;
}

polinom inmultire(polinom p1, polinom p2) {
 int g;
 polinom temp1,suma, p;
 if (p1.grad+p2.grad>MAXGRAD) {
        printf("Eroare ! \n");
        exit(1);
 }
 suma=polinom_nul();
 for (g=0; g<=p2.grad; g++)  {
        temp1=inm_termen(p2.coef[g],g, p1);
        suma=adunare(suma, temp1);
 }
 p=copie (suma);
 return p;
 }
```

La adunarea a doua polinoame p1 si p2, rezulta un polinom suma s. Acesta are gradul egal cu gradul cel mai mare dintre polinoamele p1 si p2, iar coeficientii sumei sunt obtinuti ca suma corespunzatoare a coeficientilor celor 2 polinoame operanzi.

Pentru inmultirea a doua polinoame, se defineste intai o operatie ajutatoare = inmultirea unui polinom cu un termen, prin functia `inm_termen`. Functia
`static polinom inm_termen(int coef, int grad, polinom p) ;`
este definite ca fiind de clasa `static`, pentru ca ea nu trebuie sa fie vizibila in afara fisierului `polinom.c`. De asemenea, si functia `polinom_nul()` este tot statica.

Fisierul `polinomio.c` grupeaza operatiile de intrare-iesire cu polinoame. Aceste operatii s-au grupat intr-un fisier separat de operatiile de baza cu polinoame, pentru ca definitiile lor depind mai mult de particularitatile sistemului (citirea si afisarea s-ar mai putea face din fisiere, sau utilizand biblioteci de functii de intrare-iesire cu dialoguri in mod grafic, etc), in timp ce operatiile de baza sunt aceleasi totdeauna.

```
/* polinomio.c*/

#include "polinom.h"
```

```
#include <stdio.h>

polinom  citire() {
 int g,grad,c;
 polinom p;
 for (g=0; g<MAXGRAD; g++)
      p.coef[g]=0;
 printf("Introd perechi grad-coef, terminati cu grad -1\n");
 scanf("%d %d", &g, &c);
 grad=g;
 while (g>=0) {
      p.coef[g]=c;
      if (g>grad) grad=g;
      scanf("%d %d", &g, &c);
 }
 p.grad=grad;
 return p;
}

void afisare(polinom p) {
      int g;
      for (g=p.grad; g>=0; g--)
            if (p.coef[g])
                  printf("%d * X^%d ", p.coef[g], g);
}
```

Cateva exemple de programe care fac uz de operatiile cu polinoame definite:

```
/* program_poli1.c */

#include "polinom.h"

void main(void) {
 polinom p1, p2, p3;
 p1=citire();
 p2=citire();
 p3=inmultire(p1, p2);
 afisare(p3);
}


/* program_poli2.c */

#include "polinom.h"

void main(void) {
 polinom p;
 float x=7.6;
 p=citire();
 printf(“%f ”, valoare(p, x));
 }
```