Strategii de alocare a memoriei

Fiecare din cele trei zone de date necesită o strategie de alocare proprie. Astfel:

1) **Alocarea statică** stabileşte, la compilare, memoria pentru toate obiectele de date statice.
2) **Alocarea din stivă** administrează memoria de execuție ca o stivă.
3) **Alocarea din heap** alocă şi relocă memoria când este necesar, în timpul execuției, din zona de date numită **heap**.

Aceste strategii de alocare sunt aplicate simultan, pentru aceeaşi înregistrare de acivare. Interesează, de asemenea, felul în care se realizează accesul la un nume local din codul destinație al unei proceduri.

Alocarea statică

La alocarea statică numele sunt legate de locațiile de memorie asociate la compilarea programului ⇒ nu este necesar un pachet suport de execuție. Pe parcursul execuției, legăturile nu se schimbă la fiecare activare a procedurii ⇒ numele sunt legate la aceleaşi locații de memorie. Această proprietate permite ca valorile numelor să se păstreze de la o activare la alta: când controlul revine la o procedură, valorile sunt aceleaşi care au fost când controlul a părăsit-o ultima dată.

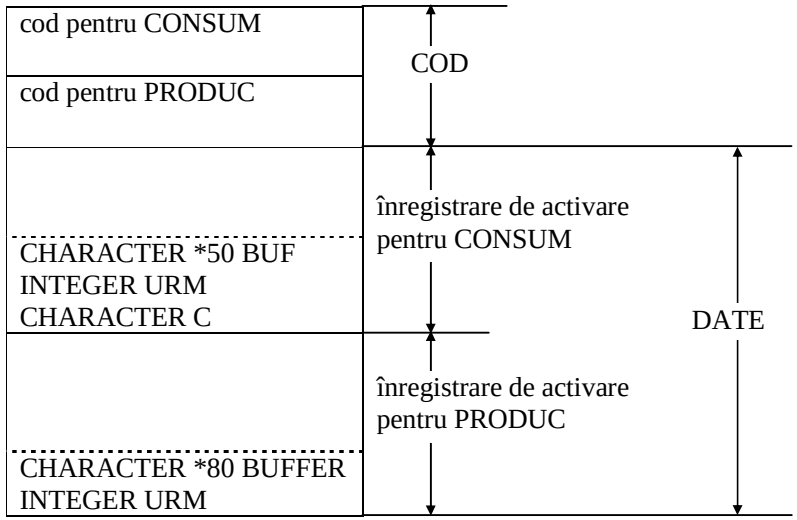
Din tipul unui nume, compilatorul determină dimensiunea locației de memorie pentru acel nume. Adresa acestei locații constă dintr-un deplasament față de un capăt al înregistrării de activare pentru procedură. Compilatorul trebuie, în final, să decidă unde se pun înregistrările de activare relativ la codul destinație precum şi una față de alta. Odată ce a fost luată această decizie, poziția fiecărei înregistrări de activare şi, implicit, a locației fiecărui nume local, a fost fixată. ⇒ La compilare, se pot completa în codul destinație adresele datelor locale. Similar, sunt de asemenea cunoscute la compilare, adresele la care trebuie salvate informații când apare un apel de procedură.

Alocare statică introduce însă şi unele limitări:

1) Dimensiunea unui obiect de date şi restricțiile asupra poziției sale în memorie, trebuie să fie cunoscute la compilare.
2) Deoarece toate activările unei proceduri utilizează aceleaşi legări pentru numele locale, apar restricții privind apelurile recursive ale procedurilor.
3) Neexistând nici un mecanism pentru alocarea memoriei la execuție, structurile de date nu pot fi create dinamic.

Un limbaj proiectat pentru a permite alocarea statică a memoriei este Fortran. Un program Fortran constă într-un program principal, subrutine şi funcții (le vom numi pe toate proceduri) ca în exemplul de mai jos (Fortran 77):

```
(1)  PROGRAM CONSUM
(2)      CHARACTER *50 BUF
(3)      INTEGER URM
(4)      CHARACTER C, PRODUC
(5)      DATA URM /1/, BUF /' '/
(6)          C=PRODUC ( )
(7)          BUF (URM : URM)=C
(8)          URM=URM+1
(9)          IF (C.NE. ' ') GOTO 6
(10)         WRITE (*,'(A)') BUF
(11)         END
(12)     CHARACTER FUNCTION PRODUC ( )
(13)         CHARACTER *80 BUFFER
(14)         INTEGER URM
(15)         SAVE BUFFER, URM
(16)         DATA URM /81/
(17)             IF (URM.GT.80) THEN
(18)             READ (*,'(A)') BUFFER
(19)             URM=1
(20)         END IF
(21)         PRODUC=BUFFER (URM : URM)
(22)         URM=URM+1
(23)         END
```



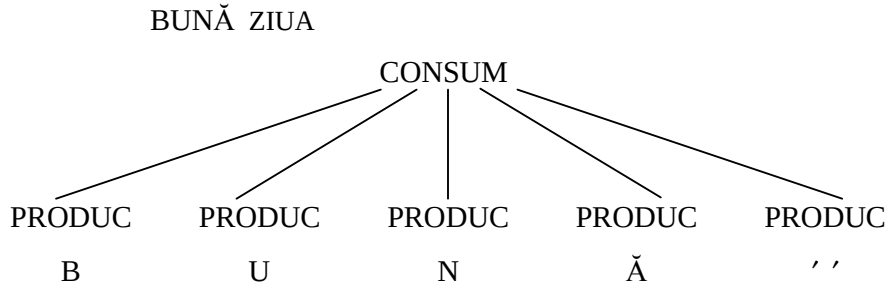
În cadrul înregistrării de activare pentru CONSUM, există spațiu pentru variabilele locale BUF, URM şi C. Locația de memorie legată la BUF conține un şir de 50 caractere. Ea este urmată de locația pentru URM în care se poate înregistra o valoare întreagă şi o locație, C, pentru un caracter.

Faptul că URM este declarat şi în PRODUC nu prezintă nici o problemă deoarece cele două variabile locale fac parte din înregistrări de activare diferite.

Dimensiunile codului executabil şi ale înregistrărilor de activare sunt cunoscute la compilare ⇒ sunt posibile şi alte organizări ale memoriei decât cea de mai sus. De exemplu, compilatorul ar putea plasa înregistrarea de activare pentru o procedură împreună cu codul pentru acea procedură. În alt caz, s-ar putea lăsa nespecificată poziția relativă a înregistrării de activare, permițând editorului de legături să lege înregistrarea de activare şi codul executabil.

Exemplu: Logica programului anterior se bazează pe faptul că valorile variabilelor locale se păstrează de la o activare la alta. Acest lucru este exprimat de instrucțiunea SAVE care specifică faptul că valoarea unei variabile locale la începutul unei activări trebuie să fie aceeaşi ca şi la sfârşitul activării precedente. Valorile inițiale ale variabilelor locale pot fi specificate utilizând instrucțiunea DATA.

Instrucțiunea READ din linia 18 a procedurii PRODUC citeşte o linie de text într-un tampon. De fiecare dată când este activată, procedura furnizează caractere succesive din acest tampon. Programul principal CONSUM are, de asemenea, un tampon în care acumulează caractere până la apariția unui spațiu. Graful de mai jos reprezintă caracterele returnate de activările lui PRODUC pentru şirul de intrare.



Ieşirea programului, tipărită în linia 10, este: BUNĂ.

Tamponul în care PRODUC citeşte linii trebuie să-şi păstreze conținutul între activări. Acest lucru este asigurat de instrucțiunea SAVE din linia 15, atât pentru BUFFER cât şi pentru URM. La primul apel al lui PRODUC, valoarea variabilei locale URM este precizată de instrucțiunea DATA din linia 16 ⇒ URM este inițializat pe 81, ceea ce asigură citirea primei linii.

Alocarea din stivă

În acest caz memoria este organizată ca o stivă iar înregistrările de activare sunt introduse şi extrase pe măsură ce încep şi respectiv se termină activările procedurilor. La fiecare apel al procedurii, memoria pentru variabilele locale este conținută în înregistrarea de activare pentru acel apel. În acest fel, variabilelor locale le corespund, la fiecare apel, locații de memorie noi. În plus, valorile variabilelor locale sunt **şterse** la încheierea activării, când înregistrarea de activare, care include locațiile de memorie, se extrage din stivă.

Se va descrie, la început, o formă de alocare din stivă în care dimensiunile tuturor înregistrărilor de activare se presupun cunoscute la compilare. Vârful stivei va fi marcat de registrul **top**. La execuție, alocarea şi relocarea unei înregistrări de activare se realizează incrementând şi respectiv decrementând **top** cu dimensiunea înregistrării: dacă procedura "q" are o înregistrare de activare de dimensiune **a**, atunci, înainte de a se executa codul lui "q", **top** este incrementat cu **a**; la închiderea apelului, când controlul revine din "q", **top** este decrementat cu **a**.

Exemplu: Să considerăm, din nou, unul din programele anterioare şi arborele său de activare. Pe măsură ce controlul parcurge acest arbore de activare, în stiva de execuție se introduc şi se extrag înregistrările de activare.

Poziția în Arborele de activare	Înregistrarea de activare în stivă	Observații
S	S a : array	a fost creată înregistrarea de activare pentru S
S C	S a : array C i : integer	a fost activat C
S C Q (1, 9)	S a : array Q(1, 9) i : integer	a fost extrasă înregistrarea de activare pentru C şi s-a introdus în stivă cea pentru Q(1, 9)
S C Q (1, 9) P (1, 9) Q (1, 3) P (1, 3) Q (1, 0)	S a : array Q(1, 9) i : integer Q(1, 3) i : integer	stiva în momentul în care controlul a revenit de la Q(1, 0) la Q(1, 3)

Liniile întrerupte din arbore merg la activări care s-au încheiat. Execuția începe cu activarea procedurii S (programul principal **sort**). Când controlul ajunge la primul apel din corpul lui S, este activată procedura C şi se introduce în stivă înregistrarea de activare pentru ea. Când controlul revine din această activare, înregistrarea este extrasă, rămânând în stivă doar înregistrarea pentru S. Apoi controlul ajunge la apelul lui Q(1, 9) şi se introduce în vârful stivei înregistrarea de activare corespunzătoare. Ori de câte ori controlul este într-o anumită activare, înregistrarea de activare pentru ea este în vârful stivei. Între penultimul şi ultimul rând din tabel s-au derulat mai multe momente, timp în care au început şi s-au încheiat activările P(1,9), P(1,3) şi Q(1,0).Înregistrările de activare corespunzătoare lor au fost introduse şi, ulterior, au fost extrase din stivă, lăsând în vârful stivei activarea pentru Q(1,3) de unde s-au declanşat apelurile P(1, 3) şi Q(1, 0).

În ceea ce priveşte accesul la datele locale într-o înregistrare de activare, acesta se poate realiza printr-un deplasament specific fiecărei date în parte raportat, de exemplu, la sfârşitul înregistrării de activare, marcată de registrul **top**. Adresele variabilelor locale pot fi însă calculate ca deplasament față de orice alt registru care indică spre o poziție fixă din înregistrarea de activare.

Secvențele de apel şi secvențele de revenire

Secvențele de apel (SA) sunt generate în codul destinație pentru a implementa apelurile de proceduri. O SA alocă înregistrarea de activare şi completează cu informații câmpurile sale. Restaurarea stării calculatorului la încheierea apelului, astfel ca să poată continua execuția procedurii apelante, este realizată de **secvența de revenire** (SR).

Secvențele de apel şi înregistrările de activare diferă chiar pentru implementările aceluiaşi limbaj. Codul corespunzător SA este deseori împărțit între procedura apelantă şi cea apelată. Nu există o împărțire exactă a sarcinilor la execuție între apelant şi apelat. Limbajul sursă, maşina destinație şi sistemul de operare impun cerințe care pot favoriza o soluție sau alta.

Un principiu care ajută atât în proiectarea SA cât şi a înregistrării de activare este plasarea la mijlocul înregistrării a câmpurilor a căror dimensiune este cunoscută devreme. De exemplu, în înregistrarea de activare generală sunt plasate la mijloc legăturile opționale de control şi acces şi starea calculatorului. Decizia dacă să se utilizeze sau nu legăturile de control şi acces este luată de proiectantul compilatorului ⇒ aceste câmpuri pot fi fixate în timpul construirii compilatorului. De asemenea, în măsura în care cantitatea de informații de stare a calculatorului este constantă, salvarea şi restaurarea stării calculatorului este aceeaşi pentru toate activările. Această uniformizare poate să fie utilizată şi de un eventual depanator de programe, pentru a descifra şi afişa conținutul stivei, în cazul apariției unei erori.

Deşi dimensiunea câmpului pentru variabilele temporare este, în final, fixată în timpul compilării, această dimensiune poate să nu fie cunoscută în partea de front-end. Generarea sau optimizarea de cod poate să reducă numărul de variabile temporare ⇒ În structura generală a înregistrării de activare acest câmp a fost plasat la sfârşit, după cel pentru date locale, astfel ca dimensiunea lui să nu afecteze deplasamentele corespunzătoare obiectelor de date locale.

Fiecare apel are propriile sale argumente care sunt evaluate de apelant şi sunt comunicate apelatului. În stiva de execuție, înregistrarea de activare a apelantului este chiar sub cea a apelatului, ca în figura de mai jos ⇒ Este avantajos să se plaseze câmpurile pentru argumente şi pentru eventuala valoare returnată imediat după înregistrarea de activare a apelantului şi în debutul celei a apelatului. În acest fel apelantul poate avea acces la aceste câmpuri utilizând deplasamentul din stivă al sfârşitului propriei înregistrări fără a cunoaşte organizarea completă a înregistrării apelatului. De altfel, nu există nici un motiv ca apelantul să cunoască datele locale sau temporare ale apelatului. O consecință benefică a acestei ascunderi a informațiilor este prelucrarea procedurilor cu un număr variabil de parametri (read şi write din Pascal, printf din C etc).

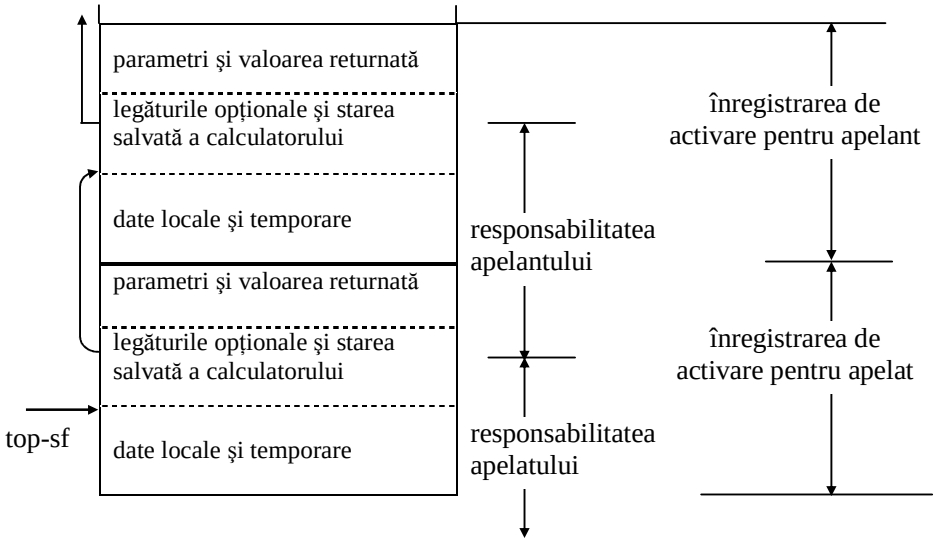


Fig. Divizarea sarcinilor între apelant şi apelat

Pascal este unul dintre puținele limbaje în care se cere ca tablourile locale unei proceduri să aibă o lungime fixă, determinabilă la compilare. În alte limbaje, dimensiunea unui tablou local poate depinde de valoarea unui argument. În acest din urmă caz, dimensiunea zonei datelor locale procedurii nu poate fi determinată până în momentul apelului (în execuție). Tehnicile pentru tratarea datelor de lungime variabilă vor fi prezentate ulterior.

În figura de mai sus, registrul **top-sf** indică la sfârşitul câmpului de stare calculatorul din înregistrarea de activare. Această poziție este cunoscută apelantului care, astfel, poate şi realiza poziționarea **top-sf**, înainte de a transfera controlul procedurii apelate. Codul pentru apelat poate realiza accesul la datele sale locale şi temporare utilizând deplasamentul față de **top-sf**.

Cu aceste precizări, SA poate consta în următoarele operații:

1) Apelantul evaluează argumentele.

2) Apelantul memorează adresa de revenire şi vechea valoare a lui **top-sf** în înregistrarea de activare a apelatului. Apelantul incrementează apoi **top-sf** la noua poziție, ca în figura de mai sus. Astfel **top-sf** trece peste datele locale şi temporare ale apelantului şi peste argumente şi starea salvată a calculatorului, corespunzătoare apelatului.

3) Apelatul salvează valoarea registrelor şi alte informații de stare.

4) Apelatul inițializează datele sale locale şi începe execuția.

O posibilă SR poate fi următoarea:

1) Apelatul plasează o eventuală valoare returnată după înregistrarea de activare a apelantului.

2) Utilizând informațiile din starea salvată, apelatul restaurează **top-sf** şi alte registre şi efectuează saltul în codul apelantului, conform adresei de revenire.

3) Deşi **top-sf** a fost decrementat, apelantul are acces şi poate utiliza valoarea returnată în evaluarea unei expresii proprii.

SA prezentată mai sus permite ca numărul de argumente pentru o procedură să depindă de apel (să fie variabil). La compilare, codul destinație al apelantului cunoaşte numărul argumentelor pe care le furnizează apelatului. În schimb, codul destinație al apelatului trebuie să fie pregătit să trateze şi alte apeluri ⇒ aşteaptă să fie apelat şi apoi examinează câmpul parametrilor. Utilizând organizarea din figura de mai sus, informațiile ce descriu parametrii trebuie plasate lângă câmpul de stare, astfel ca apelatul să le poată găsi şi identifica. De exemplu, considerând funcția standard printf din C, primul argument al lui printf specifică natura restului parametrilor ⇒ odată ce printf a localizat primul argument le poate găsi şi pe celelalte.

Date de lungime variabilă

O strategie uzuală de tratare a datelor de lungime variabilă este sugerată în figura următoare, considerând o procedură "p" care are trei tablouri locale. Locațiile de memorie pentru aceste tablouri nu sunt incluse în înregistrarea de activare pentru "p". Aceasta conține doar câte un pointer la începutul fiecărui tablou. Adresele relative ale locațiilor acestor pointeri sunt cunoscute la compilare ⇒ codul destinație poate avea acces la tablouri prin aceşti pointeri

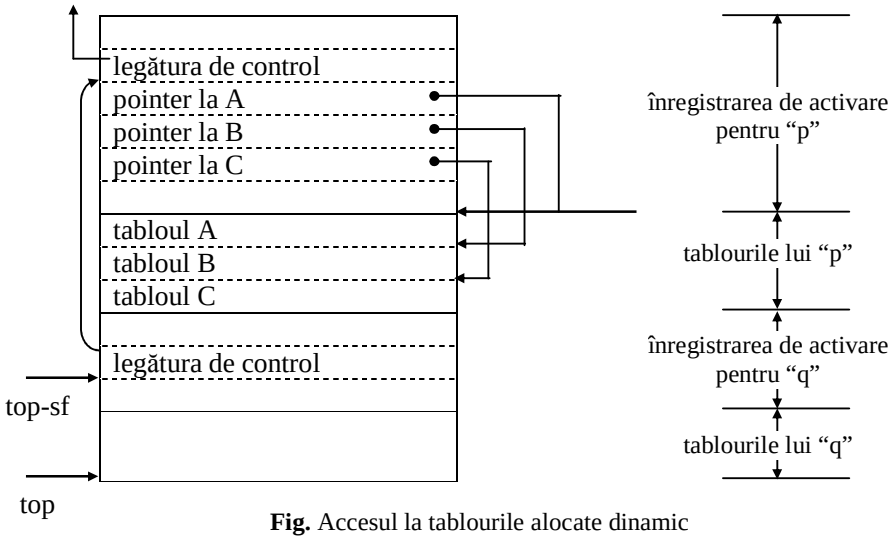


Fig. Accesul la tablourile alocate dinamic

În figura de mai sus apare şi o procedură "q", apelată din "p". Înregistrarea de activare pentru "q" începe după tablourile lui "p" iar eventualele tablouri de lungime variabilă ale lui "q" sunt plasate după aceasta.

Accesul la datele din stivă se realizează prin doi pointeri:

- **top**: marchează vârful efectiv al stivei; indică spre poziția la care va începe următoarea înregistrare de activare;
- **top-sf**: punctează datele locale.

Pentru consistență între ultima figură şi cea anterioară, presupunem că **top-sf** indică sfârşitul câmpului de stare a calculatorului din înregistrarea de activare a procedurii active (q), iar legătura opțională de control din această înregistrare marchează valoarea anterioară a lui **top-sf**: corespunzătoare situației când controlul este în procedura "p".

Pe baza dimensiunilor câmpurilor din înregistrarea de activare, cunoscute la compilare, codul pentru repoziționarea lui **top** şi **top-sf** se poate genera în timpul compilării. Astfel, la revenirea din "q", noua valoare a lui **top** este **top-sf** minus lungimea câmpurilor pentru salvarea stării calculatorului şi cel pentru parametrii lui "q". Această lungime este cunoscută la compilare, cel puțin apelantului (p). După actualizarea lui **top**, noua valoare a lui **top-sf** poate fi copiată din legătura de control a lui "q".

Referiri suspendate

Ori de câte ori se poate reloca memorie, apare problema referirilor suspendate. O **referire suspendată** este o referire din program la o zonă de memorie care a fost între timp relocată. Deoarece valoarea memoriei relocate este nedefinită, utilizarea unei referiri suspendate este o eroare logică în majoritatea limbajelor de programare. Deoarece memoria relocată poate fi ulterior alocată unei alte date, în programele cu referiri suspendate pot apărea erori dificil de depistat, cu manifestări aleatoare.

Exemplu: Să considerăm următorul program C:

```
main ()
{
    int *p
    p=eroare ();
}
int *eroare ()
{
    int i=23;
    return & i;
}
```

Procedura **eroare** returnează un pointer la memoria legată de numele local **i**. Pointerul este creat de operatorul **&** aplicat lui **i**. Când controlul revine din **eroare** la **main** memoria pentru datele locale este eliberată şi poate fi utilizată în alte scopuri. Deoarece **p** din **main** se referă la această memorie, utilizarea lui **p** este o referință suspendată.

Referințele suspendate se pot crea, evident, şi prin relocarea dinamică a memoriei, sub controlul programului.

Alocarea din heap

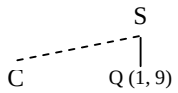
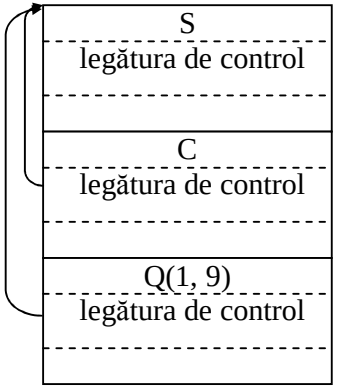
Strategia de alocare din stivă prezentată anterior nu poate fi utilizată dacă apare una din următoarele situații:

1) Valorile numelor locale trebuie păstrate când se termină activarea procedurii.
2) o activare apelată trăieşte mai mult decât apelantul. Această posibilitate nu poate să apară pentru acele limbaje în care fluxul controlului între proceduri este descris prin arbori.

În fiecare din cazurile de mai sus, relocarea memoriei nu respectă ordinea LIFO ⇒ memoria nu poate fi organizată ca stivă.

Alocarea din **heap** partiționează porțiuni de memorie contiguă, necesare atât pentru înregistrări de activare cât şi pentru alte obiecte. Porțiunile pot fi relocate în orice ordine ⇒ în timp, memoria **heap** va consta în zone alternative, libere şi utilizate(ocupate).

Diferența între alocarea din stivă şi din **heap** a înregistrărilor de activare poate fi urmărită prin comparație între tabelul anterior şi următorul:

Poziția în Arborele de activare	**Înregistrări de activare în heap**	**Observații**
		

În cazul alocării din **heap**, înregistrarea pentru o activare a procedurii C este păstrată şi după terminarea activării. ⇒ Înregistrarea pentru noua activare Q(1, 9) nu o va urma, fizic, pe cea pentru S, aşa cum s-a întâmplat în cazul anterior. Dacă, ulterior, înregistrarea lui C va fi relocată, va apărea un spațiu liber, în **heap**, între înregistrările pentru S şi Q(1, 9). Rămâne ca administratorul **heap**-ului să decidă cum să utilizeze acest spațiu.

Problema administrării eficiente a **heap**-ului este un subiect specializat în teoria structurilor de date. În general, utilizarea unui administrator de **heap** presupune regie de timp şi spațiu. Din motive de eficiență poate fi util să se trateze ca şi cazuri speciale înregistrările de dimensiuni mici sau cele de dimensiuni predictibile, astfel:

1) Pentru fiecare dimensiune de interes, se păstrează o listă de blocuri libere de acea dimensiune.
2) Dacă este posibil, se va satisface o cerere de dimensiune **s**, cu un bloc de dimensiune **s'**, unde **s'** este cea mai mică dimensiune de blocuri disponibile ≥s. În cazul relocării, blocul va fi returnat la lista din care a provenit.
3) Pentru blocuri mari de memorie, se utilizează administratorul general de **heap**.

Această abordare conduce la alocarea şi relocarea rapidă a cantităților mici de memorie, deoarece extragerea şi respectiv introducerea unui bloc dintr-o listă înlănțuită sunt operații eficiente. Pentru cantități mari de memorie, este probabil ca timpul de calcul în care se utilizează acea memorie să fie suficient de mare, astfel ca timpul necesar administratorului să poată fi neglijat în comparație cu acesta.