Clase de memorare (alocare a memoriei) în C

Clasa de memorare arată când, cum şi unde se alocă memorie pentru o variabilă.

Orice variabilă are o clasă de memorare care rezultă fie din declaraţia ei, fie implicit din locul unde este definită variabila.

Zona de memorie utilizată de un program C cuprinde 4 subzone:

Zona text: în care este păstrat codul programului
Zona de date: în care sunt alocate (păstrate) variabilele globale
Zona stivă: în care sunt alocate datele temporare (variabilele locale)
Zona heap: în care se fac alocările dinamice de memorie

Moduri de alocare a memoriei:

- **Statică:** variabile implementate în zona de date - globale

Memoria este alocată la compilare în segmentul de date din cadrul programului şi nu se mai poate modifica în cursul execuţiei. Variabilele externe, definite în afara funcţiilor, sunt implicit statice, dar pot fi declarate *static* şi variabile locale, definite în cadrul funcţiilor.

- **Auto**: variabile implementate în stivă - locale

Memoria este alocată automat, la activarea unei funcţii, în zona stivă alocată unui program şi este eliberată automat la terminarea funcţiei. Variabilele locale unui bloc (unei funcţii) şi parametrii formali sunt implicit din clasa *auto*. Memoria se alocă în stiva ataşată programului.

- **Dinamică**: variabile implementate în heap

Memoria se alocă dinamic (la execuţie) în zona *heap* ataşată programului, dar numai la cererea explicită a programatorului, prin apelarea unor funcţii de bibliotecă (*malloc, calloc, realloc*). Memoria este eliberată numai la cerere, prin apelarea funcţiei *free*

- **Register**: variabile implementate într-un registru de memorie

Clase de alocare a memoriei: Auto

Variabilele locale unui bloc (unei funcţii) şi parametrii formali sunt implicit din clasa *auto*. Durata de viaţă a acestor variabile este temporară: memoria este alocată automat, la activarea blocului/funcţiei, în zona stivă alocată programului şi este eliberată automat la ieşirea din bloc/terminarea funcţiei. Variabilele locale NU sunt iniţializate! Trebuie să le atribuim o valoare iniţială!

Clase de alocare a memoriei: Static

Memoria este alocată la compilare în segmentul de date din cadrul programului şi nu se mai poate modifica în cursul execuţiei.
Variabilele globale sunt implicit *statice* (din clasa *static*).
Pot fi declarate *static* şi variabile locale, definite în cadrul funcţiilor, folosind cuvântul cheie static. O variabilă sau o funcţie declarată (sau implicit) *static* are durata de viaţă egală cu cea a programului. In consecinţă, o variabilă *statică* declarată într-o funcţie îşi păstrează valoarea între apeluri succesive ale funcţiei, spre deosebire de variabilele *auto* care sunt realocate pe stivă la fiecare apel al funcţiei şi pornesc de fiecare dată cu valoarea primită la iniţializarea lor (sau cu o valoare imprevizibilă, dacă nu sunt iniţializate).

O sinteză legată de variabilele locale şi cele globale este:

	Variabile globale	**Variabile locale**
Alocare	Statică; la compilare	Auto; la execuţie bloc
Durata de viaţă	Cea a întregului program	Cea a blocului în care e declarată
Iniţializare	Cu zero	Nu se face automat

Alocarea dinamică a memoriei

Limbajul C permite utilizatorului să aloce date atât pe stivă (date automatice) cât şi în zone de memorie care nu aparţin stivei (date globale şi statice). Alocarea datelor pe stivă se face la execuţie şi ea nu este permanentă.

Astfel, dacă declaraţia :

tip nume ;

se utilizează în corpul unei funcţii, atunci variabila nume se alocă pe stivă la fiecare apel al funcţiei respective. La revenirea din funcţie, stiva ”se curăţă” (se readuce la starea avută înaintea apelului) şi prin aceasta variabila *nume* nu mai este alocată (devine nedefinită). O alocare de acest fel a memoriei se spune că este *dinamică.*

Pentru datele globale sau statice, memoria este alocată în fazele precedente execuţiei şi

alocarea rămâne valabilă până la terminarea execuţiei programului. De aceea pentru datele de acest fel se spune că alocarea este *statică* (nu este dinamică).

Limbajele C şi C++ oferă utilizatorului posibilitatea de a aloca dinamic memorie şi în alt mod decât cel indicat mai sus pentru datele automatice. Aceasta se realizează într-o zonă de memorie specială, distinctă de stivă. Această zonă de memorie se numeşte *memorie heap*. Ea poate fi gestionată prin funcţii standard.

Biblioteca standard a limbajului C pune la dispoziţia utilizatorului funcţii care permit alocarea de zone de memorie în timpul execuţiei programului. O astfel de zonă de memorie poate fi utilizată pentru a păstra date temporare. Zona respectivă poate fi eliberată în momentul în care nu mai sunt necesare datele care au fost păstrate în ea. Alocarea de zone de memorie şi eliberarea lor în timpul execuţiei programelor permite gestionarea optimă a memoriei de către programator. Un astfel de mijloc de gestionare a memoriei îl vom numi *alocare dinamică a memoriei.*

Funcţiile standard pentru gestiunea memoriei heap au prototipurile în fişierul *alloc.h.*

Alocarea unei zone de memorie în memoria heap se realizează cu ajutorul funcţiei malloc care are prototipul:

*void*malloc (unsigned n) ;*

Această funcţie alocă o zonă de memorie contiguă de n octeţi. Ea returnează adresa de început a zonei alocate.

Această adresă reprezintă un pointer de tip void (void *). Prin intermediul acestui pointer se pot păstra date în zona de memorie alocată în acest fel. Pentru a păstra o dată de un tip dat într-o zonă de memorie alocată prin malloc este necesar să convertim adresa returnată de funcţie spre tipul datei respective.

Exemplu:

Se cere să se aloce în memoria heap o zonă de memorie pentru a păstra n valori de tip int.

În acest scop declarăm un pointer spre tipul int: int *p ;

apoi apelăm funcţia malloc cu ajutorul expresiei de atribuire :

p = (int *) malloc(n*sizeof(int))

Valoarea returnată de funcţia malloc a fost convertită spre tipul int*, adică pointer spre tipul int.

În continuare putem pastra şi utiliza date de tip int, folosind variabila pointer p.

Funcţia malloc are ca parametru un întreg fără semn, adică acesta aparţine intervalului [0, 65.535].

În cazul în care în memoria heap nu se poate aloca o zonă de memorie contiguă de atâţia octeţi cât este valoarea parametrului de la apel, se va returna pointerul nul, adică valoarea zero. De aceea, după apelul funcţiei malloc se va testa valoarea returnată pentru a ne asigura că aceasta nu este zero.

Zonele alocate prin funcţia malloc pot fi eliberate, pentru a putea fi eventual realocate, folosind funcţia standard free. Aceasta are prototipul :

*void free (void *p) ;*

Prin apelul ei, se eliberază zona de memorie din memoria heap, spre care pointează p. (variabila p trebuie să fie obţinută printr-un apel al unei funcţii standard de alocare, cum este de exemplu malloc)

Se recomandă ca această funcţie să fie apelată de îndată ce datele dintr-o zonă de memorie heap nu mai sunt necesare, astfel zona respectivă poate fi ulterior realocată.

O altă funcţie standard utilă pentru a aloca zone de memorie în memoria heap este funţia calloc, cu prototipul :

*void *calloc (unsigned nrelem, unsigned dimelem) ;*

Funcţia alocă o zonă de memorie egală cu nrelem*dimelem octeţi. Ea returnează adresa de început a zonei de memorie alocată, adresă care reprezintă un ppinter spre void. În cazul în care nu se pot aloca nrelem*dimelem octeţi, funcţia returnează zero.

Zona de memorie alocată cu calloc, se eliberează folosind funcţia *free*.

Vectori alocaţi dinamic

Un tablou poate fi alocat dinamic printr-o secvenţă de tipul:

TIP * p;

p= (TIP *) malloc(N*sizeof(TIP));

Pointerul p va indica un bloc suficient de mare pentru a conţine N elemente de tipul TIP. În continuare, variabila p poate fi utilizată ca şi cum ar fi fost declarată ca un tablou de forma:

TIP p[N];

Avantajul alocării dinamice a unui tablou este că dimensiunea sa poate fi specificată doar în timpul execuţiei.

Un vector alocat dinamic se declară ca variabilă pointer care se iniţializează cu rezultatul funcţiei de alocare. Tipul variabilei pointer este determinat de tipul componentelor vectorului.

Exemplul 1 : Alocarea dinamică de memorie pentru un tablou de n numere întregi:

```
#include <stdlib.h>
#include <stdio.h>
int main()
{
    int n;
    int * tab;
    int i;
    printf("Introduceti numarul de elemente: \n"); scanf("%d", &n);
    if ((tab=malloc(n * sizeof(int)))==NULL)
      { printf("Eroare alocare dinamica memorie !\n");
        exit(1);
```

```
    }
  for (i=0; i<n; i++)
      printf("%d ", tab[ i ]);
  free(tab);
  return 0;
}
```

Exemplu 2: Definirea și utilizarea unui vector alocat dinamic (varianta 2)

```
#include <stdlib.h>
#include <stdio.h>
int main()
{ int n, i;
  int * a; // adresa vector alocat dinamic
  printf ("n="); scanf ("%d", &n); // dimensiune vector
  a=calloc (n,sizeof(int)); // aloca memorie pentru vector
  // sau: a=malloc (n*sizeof(int));
// citire component vector:
printf ("componente vector: \n");
for (i=0;i<n;i++)
      scanf ("%d", &a[i]); // sau scanf (“%d”, a+i);
// afisare vector:
 for (i=0;i<n;i++) printf ("%d ",a[i]);
return 0;
}
```