Gestionarea memoriei. Variabile și funcții

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

1. **Zona text**: codul programului
2. **Zona de date**: variabilele globale
3. **Zona de stivă**: date temporare (variabilele locale)
4. **Zona heap**: memoria dinamică

1. Zona de cod

Segmentul de cod (denumit și text segment) reprezintă instrucțiunile în limbaj mașină ale programului. Registrul de tip instruction pointer (IP) va referi adrese din zona de cod. Se citește instrucțiunea indicată de către IP, se decodifică și se interpretează, după care se incrementează contorul programului și se trece la următoarea instrucțiune.

Zona de cod este, de obicei, o zonă read-only pentru ca procesul să nu poată modifica propriile instrucțiuni prin folosirea greșită a unui pointer. Zona de cod este partajată între toate procesele care rulează același program.

2. Zone de date

Zonele de date conțin variabilele globale definite într-un program și variabilele de tipul read-only.

Zona de date conține

a. variabilele globale și statice inițializate la valori nenule ale unui program. De exemplu:

```
static int a = 3;
char b = 'a';
```

b. variabilele globale și statice neinițializate ale unui program. Înainte de execuția codului, acest segment este inițializat cu 0. De exemplu:

```
static int a;
char b;
```

În general aceste variabile nu vor fi prealocate în executabil, ci în momentul creării procesului.

c. informație care poate fi doar citită, nu și modificată. Aici sunt stocate constantele:

```
const int a;
const char *ptr;
```

și literalii:

```
"Hello, World!"
```

3. Zona de stivă (Zona STACK)

Stiva este o regiune dinamică în cadrul unui proces, fiind gestionată automat de compilator.

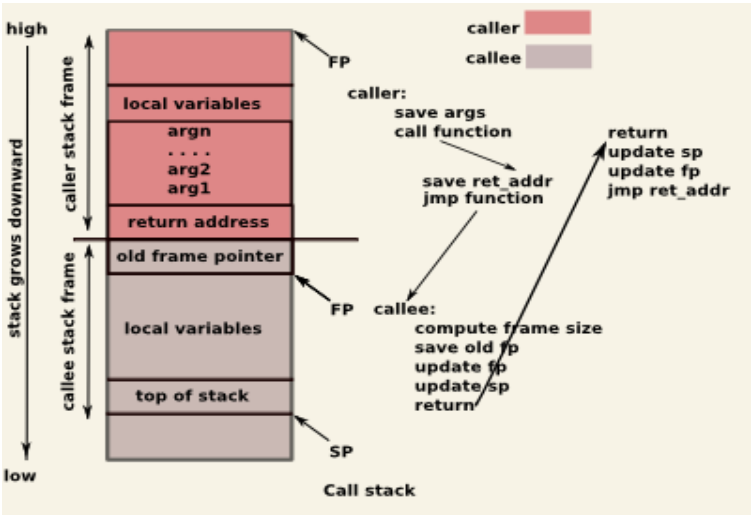
Stiva este folosită pentru a stoca *"stack frame-uri"*. Pentru fiecare apel de funcție se va crea un nou "**stack frame**".

Un "**stack frame**" conține:

- **variabile locale**
- **argumentele funcției**
- **adresa de retur**

Pe marea majoritate a arhitecturilor moderne stiva crește în jos și heap-ul crește în sus. Stiva crește la fiecare apel de funcție și scade la fiecare revenire din funcție.

În figura de mai jos este prezentată o vedere conceptuală asupra stivei în momentul apelului unei funcții.



4. HEAP-ul

Heap-ul este zona de memorie dedicată alocării dinamice a memoriei. Heap-ul este folosit pentru alocarea de regiuni de memorie a căror dimensiune este determinată la runtime.

La fel ca și stiva, heap-ul este o regiune dinamică care își modifică dimensiunea. Spre deosebire de stivă, însă, heap-ul nu este gestionat de compilator. Este de datoria programatorului să știe câtă memorie trebuie să aloce și să rețină cât a alocat și când trebuie să dealoce. Problemele frecvente în majoritatea programelor țin de pierderea referințelor la zonele alocate (memory leaks) sau referirea de zone nealocate sau insuficient alocate (accese nevalide).

În limbaje precum Java, Lisp etc. unde nu există "pointer freedom", eliberarea spațiului alocat se face automat prin intermediul unui *garbage collector*. Pe aceste sisteme se previne problema pierderii referințelor, dar încă rămâne activă problema referirii zonelor nealocate.

Variabile locale / variabile globale

- **Variabile locale**: variabile declarate în corpul unui subprogram; sunt păstrate temporar pe stiva STACK de memorie, doar atât cât se execută subprogramul, adică sunt vizibile **DOAR** în interiorul subprogramului care le-a definit; au o valoare nedefinită, reziduală, pentru aceasta **TREBUIE** inițializate;

- **Variabile globale**:sunt definite în afara oricărui subprogram; sunt vizibile în toate subprogramele definite **DUPĂ** declararea lor, adică pot fi folosite și modificate de orice subprogram; ele sunt păstrate permanent în zona/segmentul de date din memorie; la declarare ele sunt inițializate cu 0.

Observație:

Dacă sunt definite două variabile, una locală și una globală, **CU ACELAȘI NUME**, în subprogramul în care s-a definit variabila globală, compilatorul va folosi variabila locală.

Exemple de variabile globale și locale.

```
1   #include <iostream>
2   using namespace std;
3
4   int x;                    Variabile
5   char s='A';               globale
6   float z[20];
7
8   int doi()
9   {
10      int x = 2;  Variabila locala
11      return x;
12  }
13
14  int main()
15  {
16      int a;          Variabile locale
17      int b = 5;
18      a = b*doi();
19      cout<< a;
20      return 0;
21  }
```

Durata de viaţă și domeniu de vizibilitate al variabilelor

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