# Chapitre 13 : La mémoire en C

## Construction et maintenance de logiciels

Guy Francoeur

basé sur les travaux d'Alexandre Blondin Massé, professeur

5 septembre 2019

UQÀM | Département d'informatique

# Table des matières

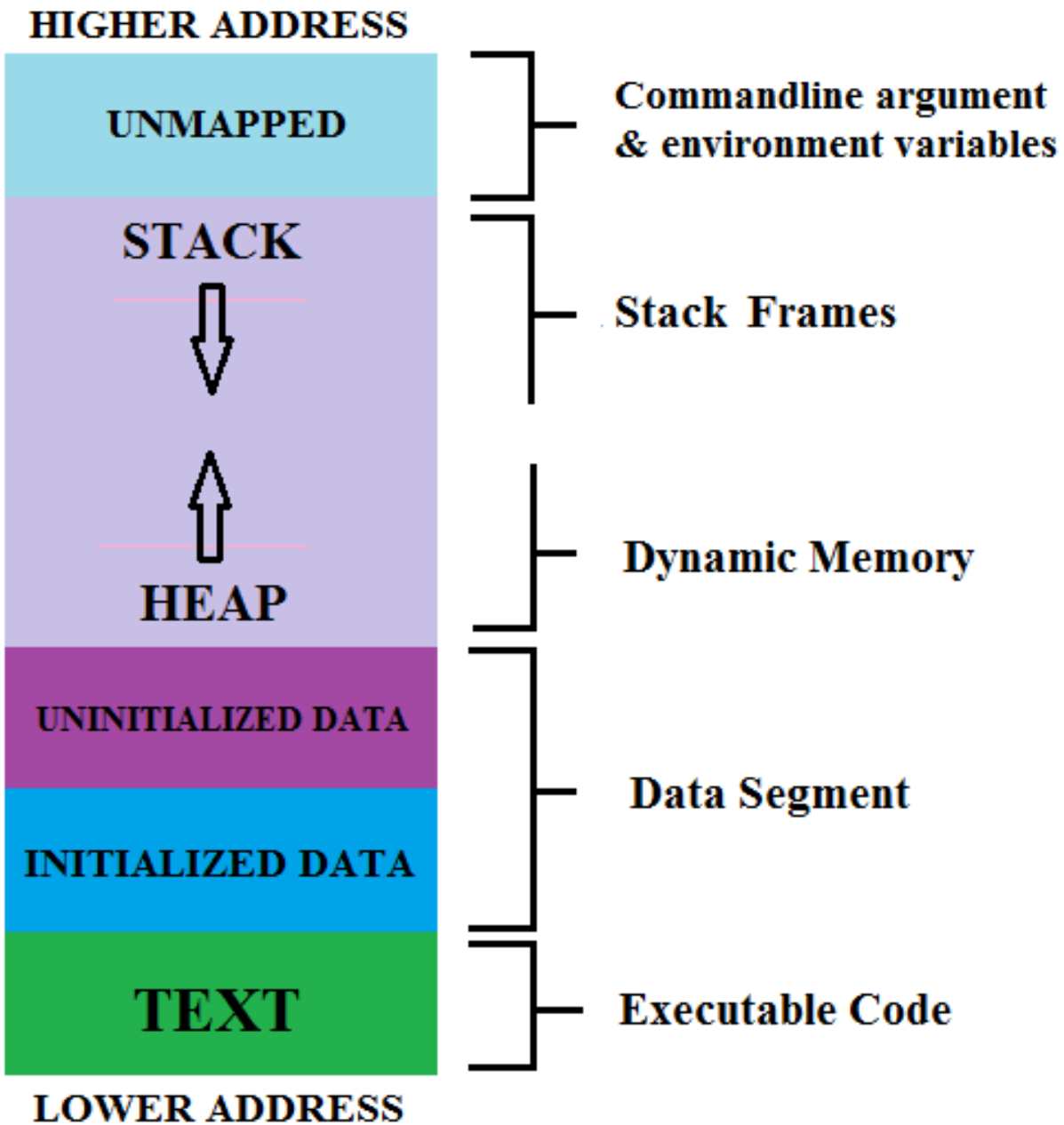
# Table des matières

# Avis au lecteur

Dans ce chapitre nous utiliserons plusieurs termes en anglais. Ceci afin de présenter et préserver l'exactitude des termes d'usage courant. Il est possible que certains mots soient traduits dans les versions futures.

# Table des matières

# Introduction des segments de la mémoire



# L'organisation de la mémoire en C

- *Text segment* : Le code binaire de l'exécutable;
- *Data segment* : Data est divisé en deux sections ( Initialized data et Uninitialized data);
  - Initialized : variables globales, static initialisé;
  - Uninitialized : variables globales et static non implicitement initialisé dans le code source ou initialisé à zéro;
- *Heap segment* : les allocations de mémoire dynamique;
- *Stack segment* : les variables locales et les appels de fonctions;
- *Unmapped or reserved* : les arguments de la ligne de commande;

# Table des matières

# Text segment

- Contient le code exécutable (code machine) de votre programme;
- Toutes les instructions y sont représentées, incluant les appels vers les fonctions locales (*user defined*) et les appels (fonctions) systèmes;
- D'ordinaire le segment Text est partagé, donc une seule copie de l'exécutable sera chargée en mémoire;
- Le segment Text une fois le programme chargé devient en mode *READ-ONLY*, afin de prévenir les altérations du logiciel;

# Table des matières

# Data segment

- Initialisé
  - variable globale;
  - variable statique;
  - ... initialisé avec une valeur autre que zéro.
- Non-initialisé
  - variable globale;
  - variable statique (static);
  - variable qui n'est pas explicitement initialisé dans le code;
  - variable qui est initialisé avec la valeur zéro.

# Table des matières

# Heap segment

- Les allocations dynamiques sont envoyés dans le **tas**;
- Usage de **malloc, calloc, realloc** en C, **new** en C++;
- La libération avec **free** en C et **delete** C++;
- Source fréquentes fuites (*leak*) de mémoire;

# Table des matières

# Stack segment

- Contient les variables locales;
- Lorsqu'une fonction est appelée un *stack frame* est créé;
- Lorsque la fonction est de retour, le *stack frame* est détruit ainsi que toutes les variables locales;
- Le *stack frame* (sous section du *stack segment*) contient des (*data*) informations telles que:
  - L'adresse de retour;
  - Les arguments;
  - Les variables locales;
  - Ainsi que d'autres informations requises par la fonction.

# Stack segment —> stack frame

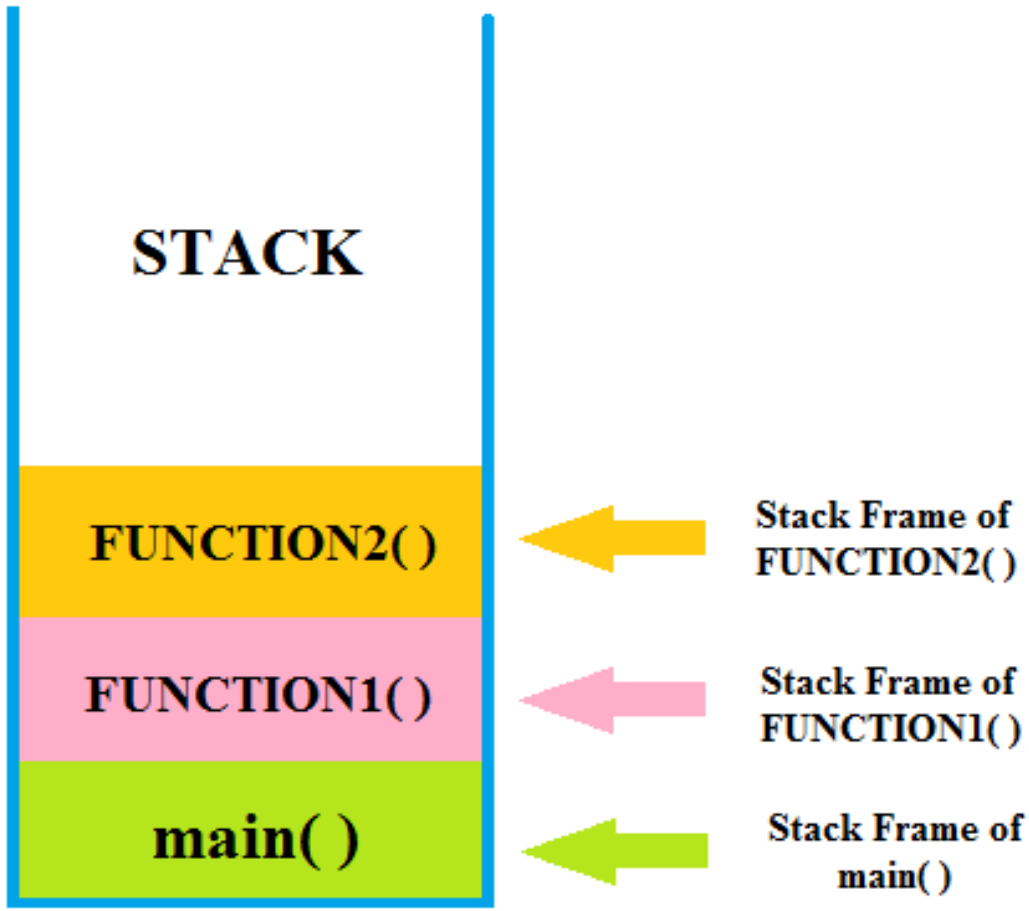



Figure: stack frame

# Table des matières

# Reserved segement

- Contient les arguments (options) de la ligne de commande;
- Le bornage des zones (segment) de mémoire pour l'exécution de votre programme;

# Exemple C

```c
//memoire.c
#include <unistd.h> //getpid
#include <stdio.h> //printf, sprintf
#include <stdlib.h> //malloc, system

int giGlobale;
char* gcDynamic;
typedef unsigned long UL;
int main() {
  int iLocale;
  gcDynamic = (char *) malloc(1024L * 1024L * 2L); /* 2mo */
  printf("PID = %d\n", getpid());
  printf("adresse de : giGlobale = %8lx\n", (UL) &giGlobale);
  printf("adresse de : iLocale   = %8lx\n", (UL) &iLocale);
  printf("adresse de : gcDynamic = %8lx\n", (UL)  gcDynamic);
  printf("adresse de : une_fonction = %8lx\n", (UL) &main);
  printf("adresse de : printf       = %8lx\n", (UL) &printf);
  /* afficher la carte mmoire */
  sprintf(gcDynamic, "cat /proc/%d/maps", getpid());
  system(gcDynamic);
  free(gcDynamic);
  return 0;
}
```

# Les segments de la mémoire