1. Phân vùng nhớ MCU

- Gồm 5 phân vùng nhớ chính:

***1. Text segment (text).***

- Là nơi chưa mã lệnh thực thi của chương trình.

- Quyền truy cập: Only read

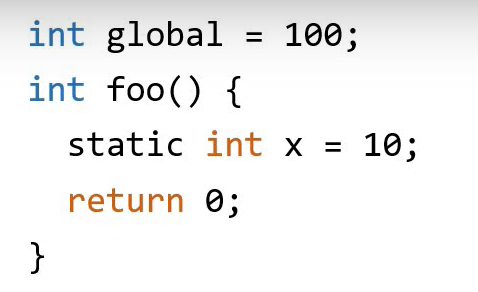
- Chứa những khai báo hằng số trong chương trình(.rolata – Vùng chỉ read)

***2. Intialized data segment (data) DS:*** *Lưu biến : Global Variables & Static Variables*

- Là nơi chứa biến toàn cục hoặc biến static có giá trị khởi tạo khác 0 ( hiểu là được gán giá trị). Biến này có thể thay đổi được trong lúc chạy chương trình.

- Quyền truy cập: Read – Write.

- Được giải phóng khi kết thúc chương trình.

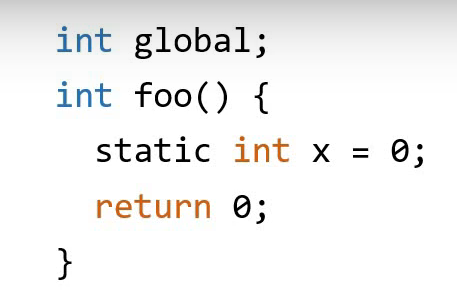


***3. Uninitiazed data segment (BSS):****Lưu biến : Global Variables & Static Variables*

- Chứa biến toàn cục hoặc biến static không khởi tạo hoặc giá trị khởi tạo bằng 0.

- Quyền truy cập: Read – Write.

- Được giải phóng khi kết thúc chương trình.



***4. Stack:***

- Được dùng để cấp phát bộ nhớ cho tham số của các hàm (function parameters) và biến cục bộ (local variables) chứa các funstion frame trong quá trình mà ct xử lý.

- Có cấu trúc: LIFO Structure ( Last In First Out) . Chúng ta chỉ có thể thao tác với phần tử nằm trên cùng (ngoài cùng),Có nghĩa là phần tử nào được thêm vào mảng sau cùng thì sẽ được lấy ra đầu tiên.

- Hàm sẽ được giải phóng khi ra khỏi block code/ hàm

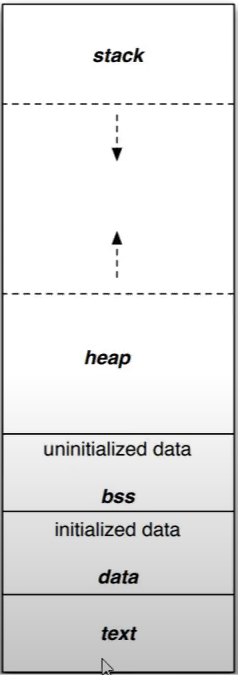
***5. Heap***.

- Được sử dụng để Cấp phát bộ nhớ động (Dynamic memmory Allocation) thông qua gọi hàm: malloc, new

- Sử dụng hàm realloc() để thay đổi (phân bổ lại) kích thước bộ nhớ đã cấp phát trong khi chạy chương trình.

- Vùng nhớ này sẽ **grows upward** khi cấp phát do vậy, Để giải phóng bộ nhớ đã cấp phát khi không sử dụng ta dùng hàm: free, delete.

-



**\* So sánh phân vùng Head và Stack:**

- Điểm giống nhau:

+ Bộ nhớ Heap và bộ nhớ Stack bản chất đều cùng là vùng nhớ được tạo ra và lưu trữ trong RAM khi chương trình được thực thi.

- Điểm khác nhau:

|  |  |
| --- | --- |
| Heap | Stack |
| + Bộ nhớ Heap được dùng để lưu trữ vùng nhớ cho những biến con trỏ được cấp phát động bởi các hàm malloc - calloc - realloc (trong C). Truy cập rất lâu | + Bộ nhớ Stack được dùng để lưu trữ các biến cục bộ trong hàm, tham số truyền vào...  Truy cập vào bộ nhớ này rất nhanh và được thực thi khi chương trình được biên dịch. |
| + Nếu bạn liên tục cấp phát vùng nhớ mà không giải phóng thì sẽ bị lỗi tràn vùng nhớ Heap (Heap overflow). | + Vùng nhớ Stack được quản lý bởi hệ điều hành, dữ liệu được lưu trong Stack sẽ tự động hủy khi hàm thực hiện xong công việc của mình. |
| + Kích thước của bộ nhớ Heap là không cố định, có thể tăng giảm do đó đáp ứng được nhu cầu lưu trữ dữ liệu của chương trình. | + Kích thước của bộ nhớ Stack là cố định, tùy thuộc vào từng hệ điều hành, bởi vì bộ nhớ Stack cố định nên nếu chương trình bạn sử dụng quá nhiều bộ nhớ vượt quá khả năng lưu trữ của Stack chắc chắn sẽ xảy ra tình trạng tràn bộ nhớ Stack (Stack overflow) |
| + Vùng nhớ Heap được quản lý bởi lập trình viên | + Vùng nhớ Stack được quản lý bởi hệ điều hành, dữ liệu được lưu trong Stack sẽ tự động hủy khi hàm thực hiện xong công việc của mình. |

2. Funstion, Inline, Marco:

1. Funstion:

- Khi thấy hàm được gọi, compiler sẽ phải lưu con trỏ chương trình PC hiện tại vào stack; chuyển PC tới hàm được gọi, thực hiện hàm đó xong và lấy kết quả trả về; sau đó quay lại vị trí ban đầu trong stack trước khi gọi hàm và tiếp tục thực hiện chương trình.

- Như có thể thấy, các này khiến chương trình tốn thời gian hơn là chỉ cần thay thế đoạn code đã được compile (cách của inline function).

2. Inline

- Được xử lý bởi compiler

- Được khai báo với từ khóa inline

- Khi compiler thấy bất kỳ chỗ nào xuất hiện inline function, nó sẽ thay thế chỗ đó bởi định nghĩa của hàm đã được compile tương ứng. –> Phần được thay thế không phải code mà là đoạn code đã được compile.

3. Macro:

- Được xử lý bởi preprocessor

- Thay thế đoạn code được khai báo macro vào bất cứ chỗ nào xuất hiện macro đó

VD: #define SUM(a,b)     (a+b)

- Preprocessor khi gặp bất kỳ lời gọi SUM(first+last) nào thì thay ngay bằng (first+last)

\* So sánh Funstion, Macro và Inline

- Macro đơn giản là chỉ thay thế đoạn code macro vào chỗ được gọi trước khi được biên dịch

- Inline thay thế đoạn mã code đã được biên dịch vào chỗ được gọi

- Hàm bình thường phải tạo một function call, lưu địa chỉ trước khi gọi hàm vào stack sau đó mới thực hiện hàm và sau cùng là quay trở về địa chỉ trên stack trước khi gọi hàm và thực hiện tiếp chương trình

- Macro khiến code trở nên dài hơn rất nhiều so với bình thường nhưng thời gian chạy nhanh.

- Hàm inline cũng khiến code dài hơn, tuy nhiên nó làm giảm thời gian chạy chương trình

Hàm bình thường sẽ phải gọi function call nên tốn thời gian hơn inline function nhưng code ngắn gọn hơn.