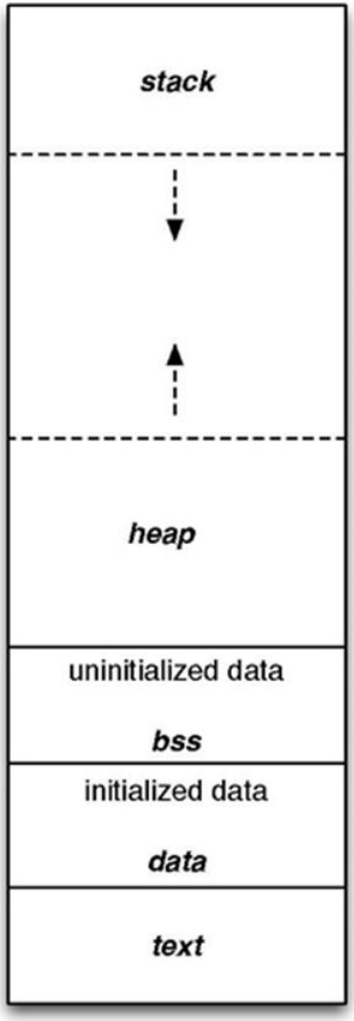
Bài 8: Memory Layout

Nguyễn Hoàng Anh

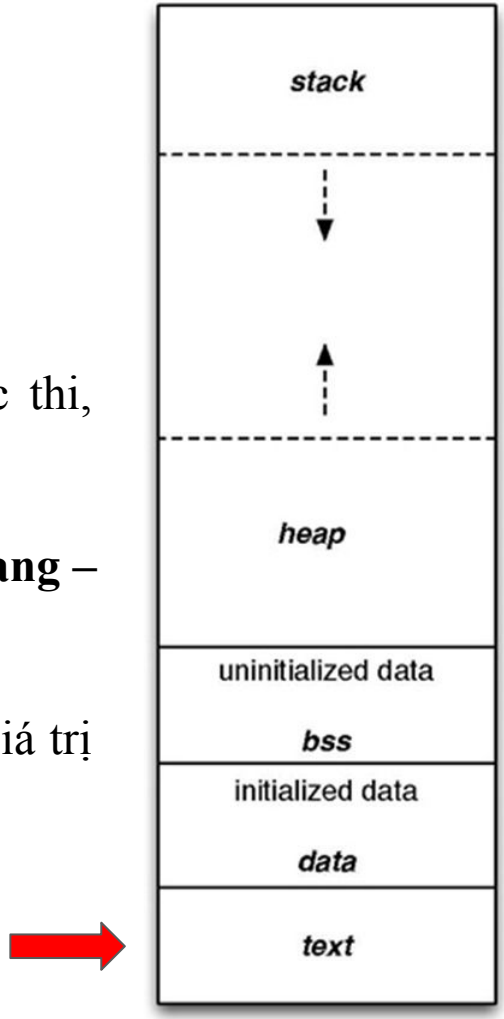
Memory layout

Chương trình main.exe (trên window), main.hex (nạp vào vi điều khiển) được lưu ở bộ nhớ **SSD** hoặc **FLASH**. Khi nhấn run chương trình trên window (cấp nguồn cho vi điều khiển) thì những chương trình này sẽ được copy vào bộ nhớ RAM để thực thi.



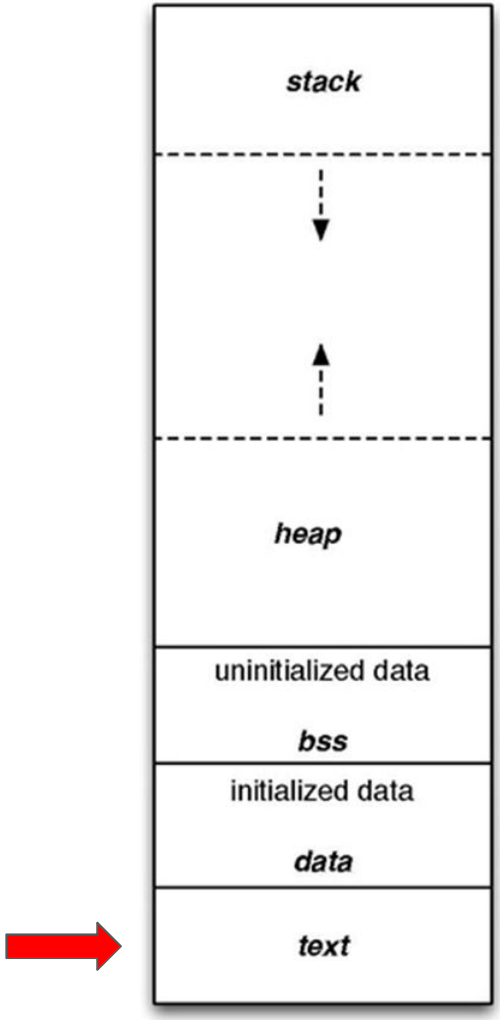
Text segment (Code segment)

- Mã máy: chứa tập hợp các lệnh thực thi.
- Quyền truy cập: Text Segment thường có quyền đọc và thực thi, nhưng không có quyền ghi.
- **Lưu hằng số toàn cục (const), chuỗi hằng - string literal (Clang – macOS, windows - minGW)**
- Tất cả các biến lưu ở phần vùng Text đều không thể thay đổi giá trị mà chỉ được đọc.



Text segment

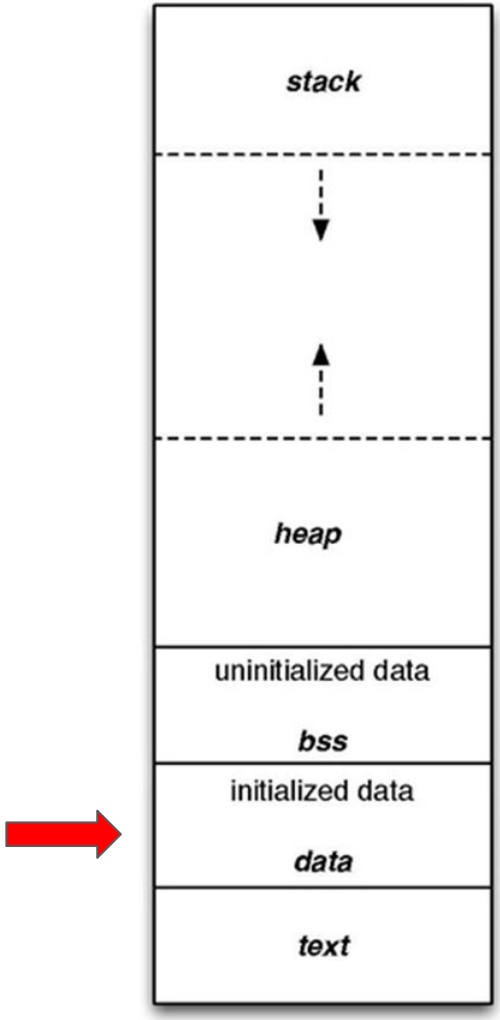
```c
#include <stdio.h>

const int a = 10;
char arr[] = "Hello";
char *arr1 = "Hello";
int b = 0;
int *ptr = &b;

int main() {


    printf("a: %d\n", a);

    arr[3] = 'W';
    printf("arr: %s", arr);

    arr1[3] = 'E';
    printf("arr1: %s", arr1);

    return 0;
```



Data segment

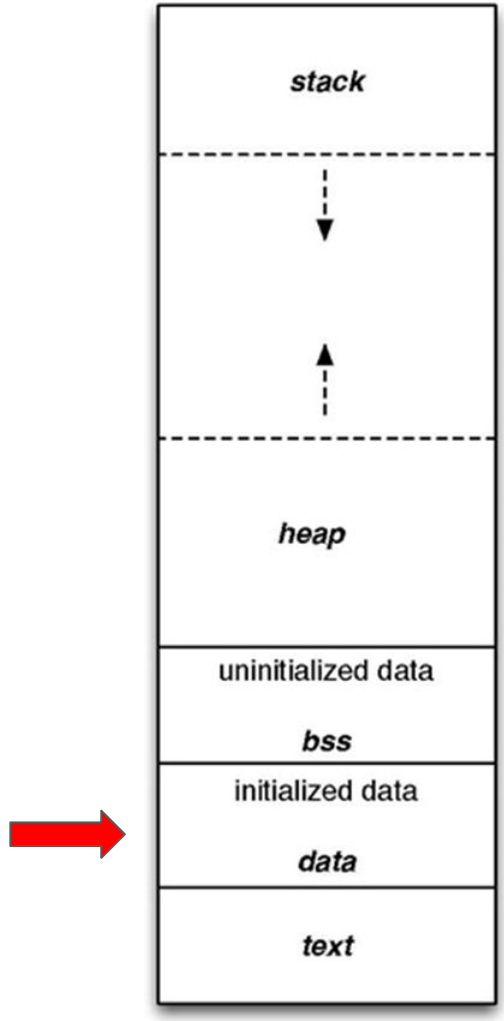
- Initialized Data Segment (Dữ liệu Đã Khởi Tạo):

- Chứa các biến toàn cục được khởi tạo với **giá trị khác 0**.
- Chứa các biến **static (global + local)** được khởi tạo với giá trị khác 0.
- Quyền truy cập là đọc và ghi, tức là có thể đọc và thay đổi giá trị của biến .
- Tất cả các biến sẽ được thu hồi sau khi chương trình kết thúc.



Data segment

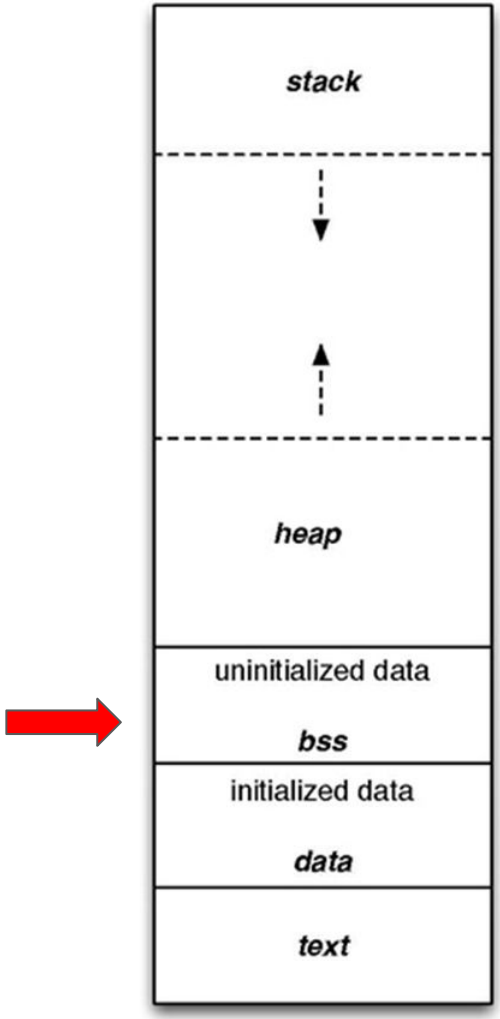
```c
#include <stdio.h>

int a = 10;
double d = 20.5;

static int var = 5;

void test()
{
    static int local = 10;
}


int main(int argc, char const *argv[])
{
    a = 15;
    d = 25.7;
    var = 12;
    printf("a: %d\n", a);
    printf("d: %f\n", d);
```



Bss segment

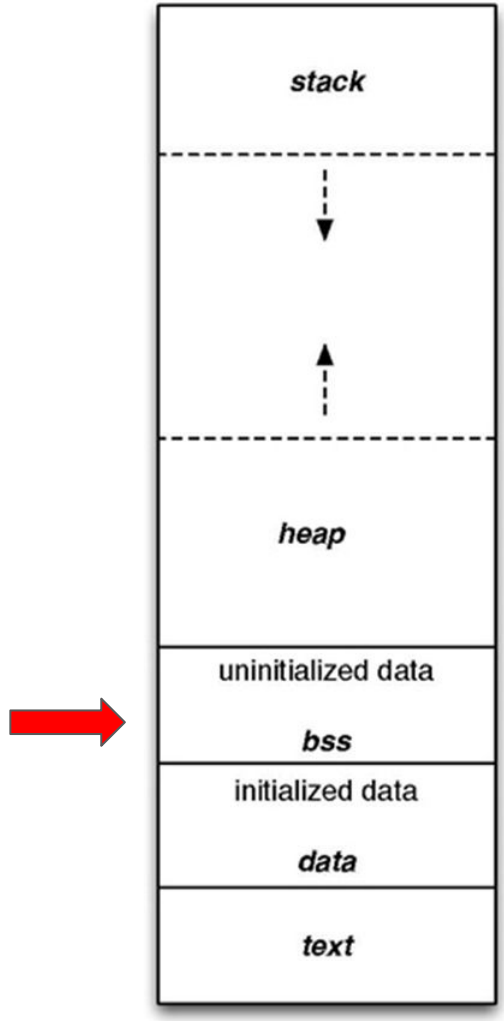
- Uninitialized Data Segment (Dữ liệu Chưa Khởi Tạo):

- Chứa các biến toàn cục khởi tạo với giá trị **bằng 0** hoặc **không gán giá trị**.
- Chứa các biến static với giá trị khởi tạo bằng 0 hoặc không gán giá trị.
- Quyền truy cập là đọc và ghi, tức là có thể đọc và thay đổi giá trị của biến .
- Tất cả các biến sẽ được thu hồi sau khi chương trình kết thúc.



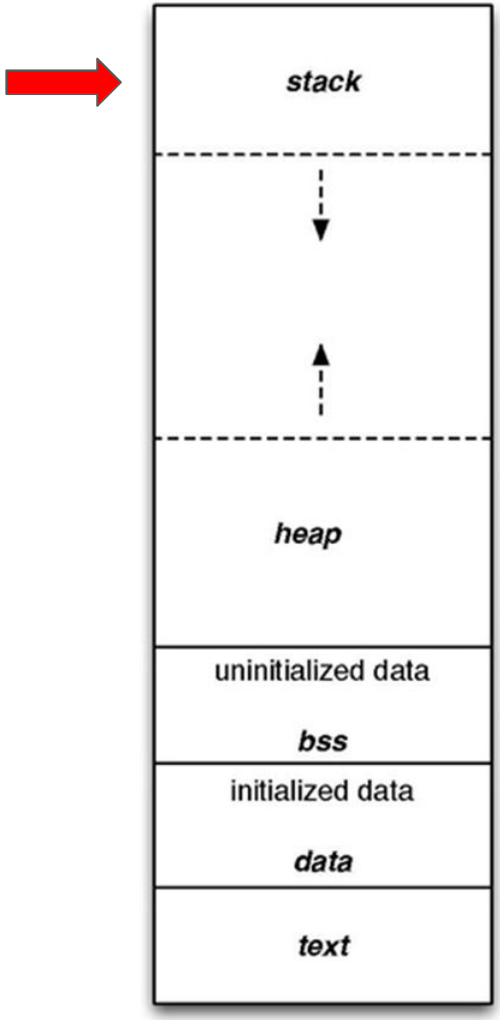
Bss segment

```c
#include <stdio.h>


typedef struct
{
    int x;
    int y;
} Point_Data;


int a = 0;
int b;

static int global = 0;
static int global_2;

static Point_Data p1 = {0,0};


void test()
```



Stack

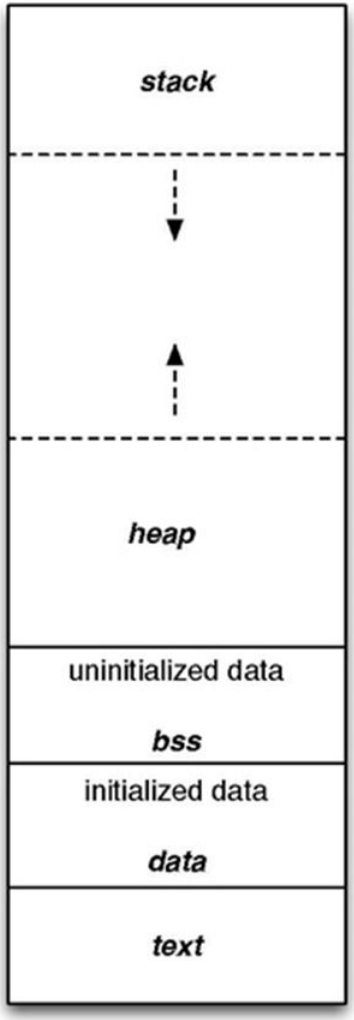
- Chứa các biến cục bộ (trừ static cục bộ), tham số truyền vào.
- Hằng số cục bộ, có thể thay đổi thông qua con trỏ.
- Quyền truy cập: đọc và ghi, nghĩa là có thể đọc và thay đổi giá trị của biến trong suốt thời gian chương trình chạy.
- Sau khi ra khỏi hàm, **tự động** thu hồi vùng nhớ.



Stack

```c
#include <stdio.h>


void test()
{
    int test = 0;
    test = 5;
    printf("test: %d\n",test);
}

int sum(int a, int b)
{
    int c = a + b;
    printf("sum: %d\n",c);
    return c;
}
```

```c
int main() {
```



Heap

uint16_t arr[5] = {2,3,5,6,8}

0x01	0x03	0x05	0x07	0x09
2	3	5	6	8

uint32_t arr[5] = {2,3,5,6,8}

0x01	0x05	0x09	0x0D	0x11
2	3	5	6	8

Heap

```c
#include <stdio.h>
#include <stdint.h>

uint32_t arr[] = {2,3,5,6,8};


int main() {

    for (int i = 0; i < 5; i++)
    {
        printf("Address: %p\n", arr +i);
        printf("Value: %d\n", *(arr+i));
    }


    return 0;
}
```

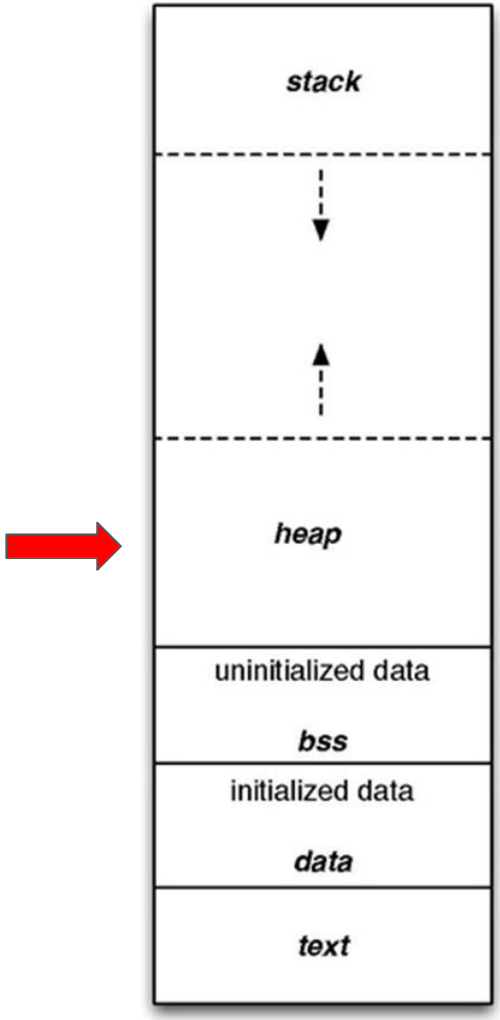
Heap

Đặt vấn đề:
Viết chương trình yêu cầu người dùng nhập tên, sau đó hiển thị tên vừa nhập

Heap

- Cấp phát động:

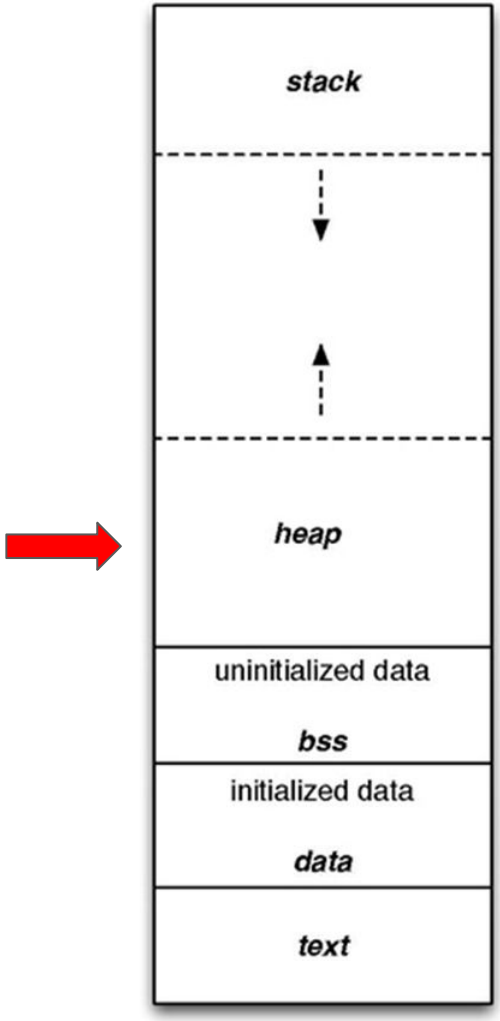
- Heap được sử dụng để cấp phát bộ nhớ động trong quá trình thực thi của chương trình.
- Điều này cho phép chương trình tạo ra và giải phóng bộ nhớ theo nhu cầu, thích ứng với sự biến đổi của dữ liệu trong quá trình chạy.
- Các hàm như malloc(), calloc(), realloc(), và free() được sử dụng để cấp phát và giải phóng bộ nhớ trên heap.



Heap

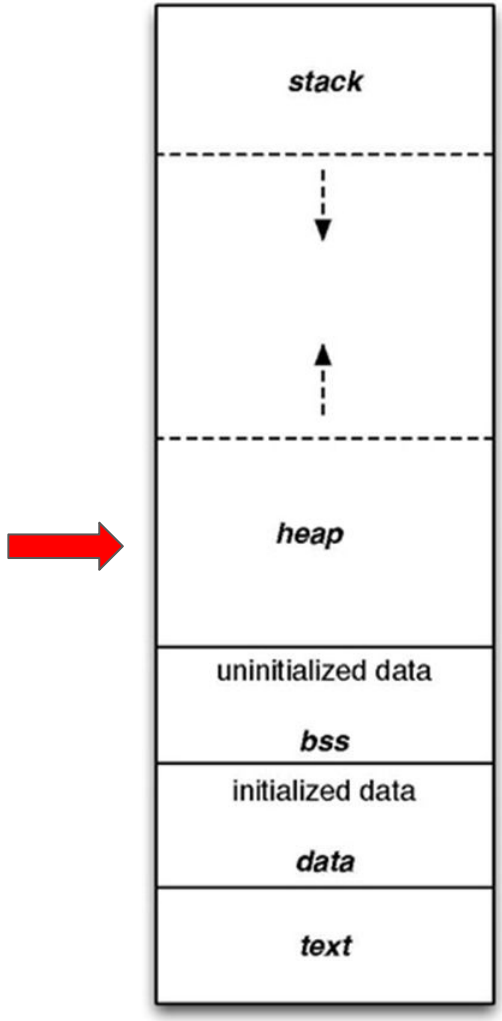
- malloc():

- Tham số truyền vào: kích thước mong muốn (byte)
- Giá trị trả về: con trỏ void



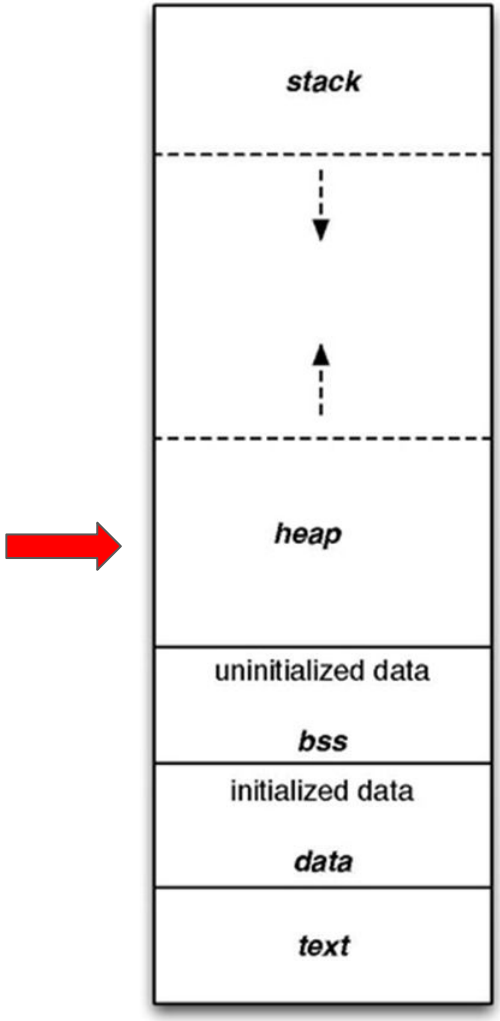
Heap

```c
#include <stdlib.h>

int main() {
    int *arr_malloc, *arr_calloc;
    size_t size = 5;

    // Sử dụng malloc
    arr_malloc = (int*)malloc(size * sizeof(int));

    // Sử dụng calloc
    arr_calloc = (int*)calloc(size, sizeof(int));

    // ...

    // Giải phóng bộ nhớ
    free(arr_malloc);
    free(arr_calloc);

    return 0;
}
```



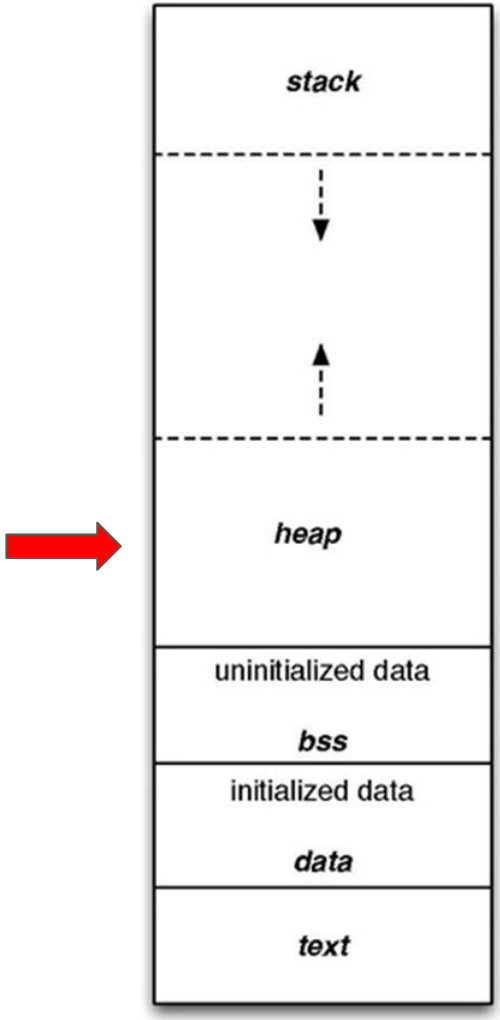
Heap

```c
#include <stdio.h>
#include <stdlib.h>


int main(int argc, char const *argv[])
{
    int soluongkytu = 0;

    char* ten = (char*) malloc(sizeof(char) * soluongkytu);



    for (int i = 0; i < 3; i++)
    {
        printf("Nhap so luong ky tu trong ten: \n");
        scanf("%d", &soluongkytu);
        ten = realloc(ten, sizeof(char) * soluongkytu);
        printf("Nhap ten cua ban: \n");
        scanf("%s", ten);

        printf("Hello %s\n", ten);
```



Heap

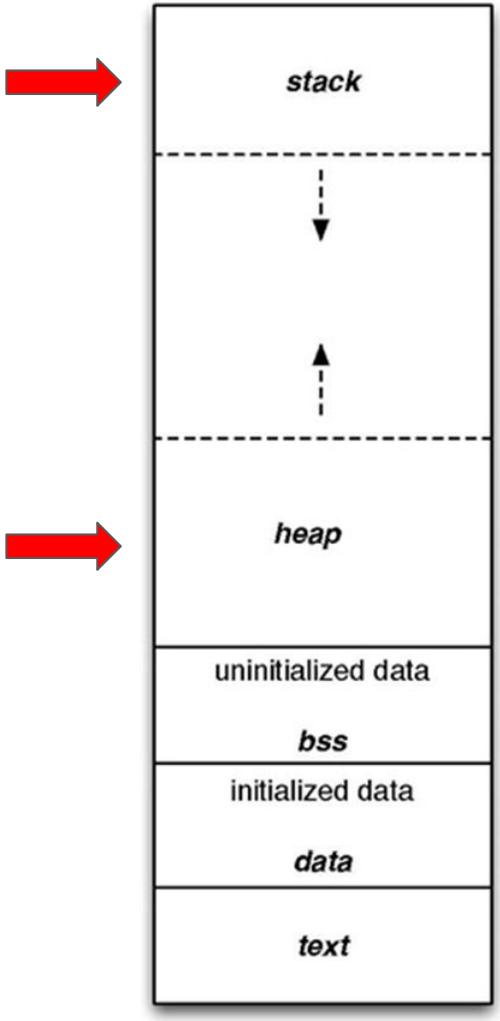
- Quyền truy cập: có quyền đọc và ghi, nghĩa là có thể đọc và thay đổi giá trị của biến trong suốt thời gian chương trình chạy.



Stack và Heap

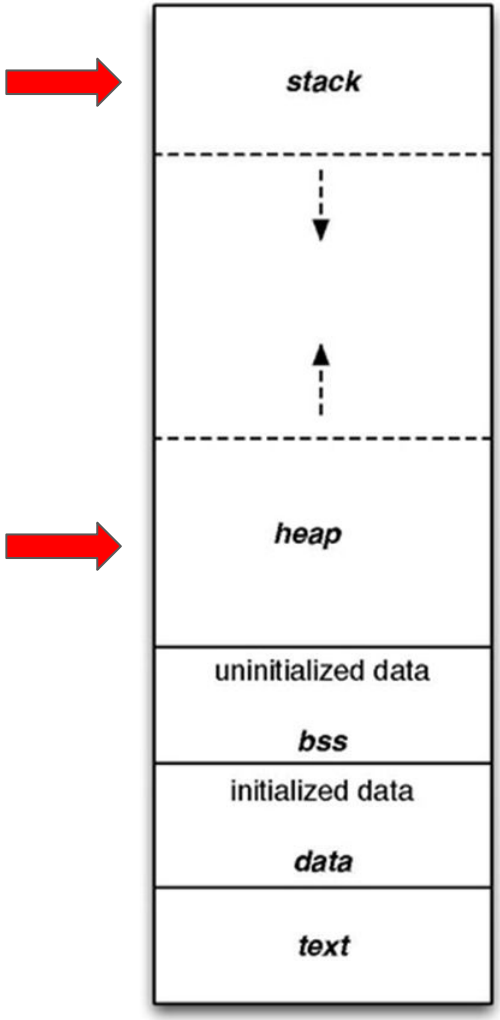
Bộ nhớ Stack được dùng để lưu trữ các biến cục bộ trong hàm, tham số truyền vào... Truy cập vào bộ nhớ này rất nhanh và được thực thi khi chương trình được biên dịch.

Bộ nhớ Heap được dùng để lưu trữ vùng nhớ cho những biến được cấp phát động bởi các hàm *malloc* - *calloc* - *realloc* (trong C).



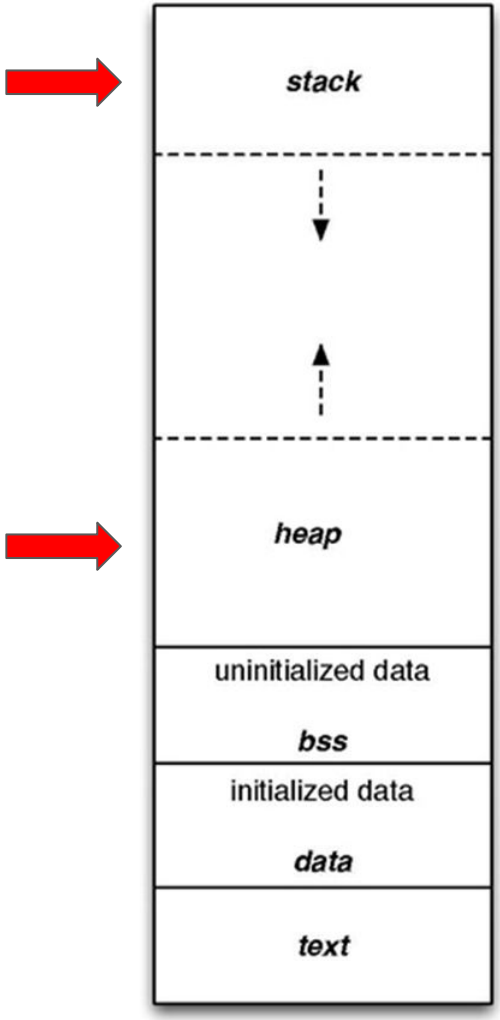
Stack và Heap

- Stack: vùng nhớ Stack được quản lý bởi hệ điều hành, dữ liệu được lưu trong Stack sẽ tự động giải phóng khi hàm thực hiện xong công việc của mình.
- Heap: Vùng nhớ Heap được quản lý bởi lập trình viên (trong C hoặc C++), dữ liệu trong Heap sẽ không bị hủy khi hàm thực hiện xong, điều đó có nghĩa bạn phải tự tay giải phóng vùng nhớ bằng câu lệnh free (trong C), và delete hoặc delete [] (trong C++), nếu không sẽ xảy ra hiện tượng rò rỉ bộ nhớ.



Stack và Heap

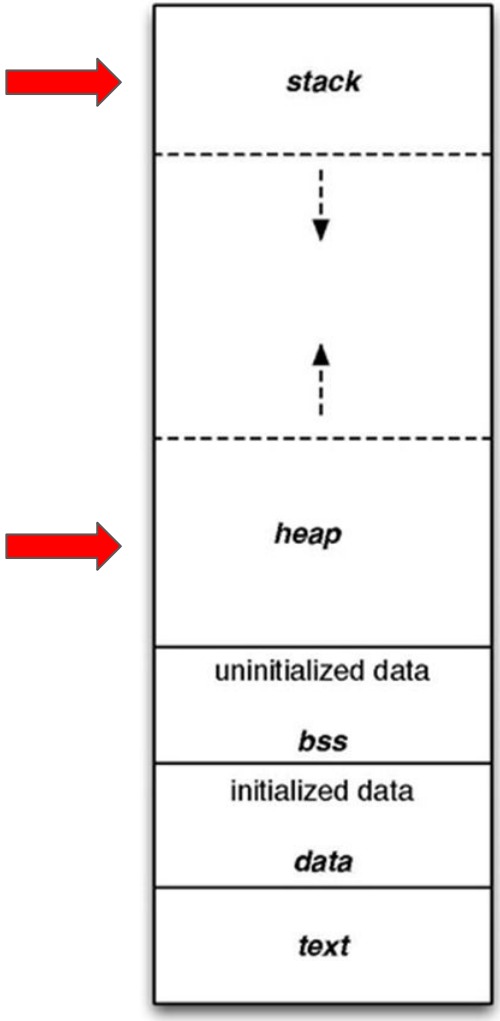
```c
#include <stdio.h>
#include <stdlib.h>

void test1(){
    int array[3];
    for (int i = 0; i < 3; i++){
        printf("address of array[%d]: %p\n", i, (array+i));
    }
    printf("----------------------\n");
}

void test2(){
    int *array = (int*)malloc(3*sizeof(int));
    for (int i = 0; i < 3; i++){
        printf("address of array[%d]: %p\n", i, (array+i));
    }
    printf("----------------------\n");
    free(array);
}
```



Stack và Heap

- Stack: 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), các trường hợp xảy ra như bạn khởi tạo quá nhiều biến cục bộ, hàm đệ quy vô hạn,...

```
int foo(int x){
    printf("De quy khong gioi han\n");
    return foo(x);
}
```



Stack và Heap

- Heap: Nếu bạn liên tục cấp phát vùng nhớ mà không giải phóng thì sẽ bị lỗi tràn vùng nhớ Heap (Heap overflow). Nếu bạn khởi tạo một vùng nhớ quá lớn mà vùng nhớ Heap không thể lưu trữ một lần được sẽ bị lỗi khởi tạo vùng nhớ Heap thất bại.

```
int *A = (int *)malloc(18446744073709551615);
```