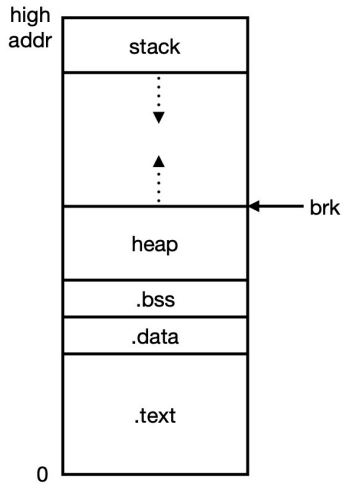
Implementação de Linguagens
– gestão do "heap" sem "garbage collection" –

---

1. O exercício aqui proposto tem como objectivo mostrar uma implementação possível de um gestor para o "heap"de um processo. O enunciado e o código é baseado no excelente artigo: Memory Allocators 101 - write a simple memory allocator - por Arjun Sreedharan. O código original para o projecto pode ser visto no repositório `github` do autor. No entanto, sugiro que use uma versão ligeiramente editada fornecida no Moodle.

O espaço de endereçamento de um processo, com os segmentos que o compõem, está representado na figura seguinte.



Durante a execução de um processo, o espaço reservado para a "stack" é utilizado para manter informação sobre o encadeamento das chamadas a funções e os respectivos ambientes locais (argumentos e variáveis locais). Por outro lado, o "heap" é utilizado para manter estruturas de dados que podem persistir entre chamadas a funções. Estas estruturas de dados são acessíveis via referências mantidas em variáveis locais na "stack" ou globais nos segmentos "data" ou "bss".

A linguagem C não usa "garbage collection". O "heap" é gerido pelo programador usando o seguinte conjunto de funções da `libc`:

```
void*  malloc(size_t)
void*  calloc(size_t, size_t)
void*  realloc(void*, size_t)
void   free(void*)
```

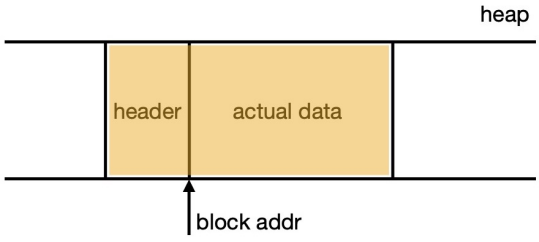
A reserva de memória é feita explicitamente pelo programador, usando as funções `malloc()`, `calloc()` e `realloc()`. Os blocos de memória assim reservados têm de ser explicitamente libertados pelo programador utilizando a função `free()`. Cada um destes blocos é então adicionado a uma lista de blocos que podem ser reutilizados pelas funções que reservam espaço, em vez de reservar espaço extra no topo do "heap", fazendo avançar o apontador `brk`.

Note que os blocos de memória reservados têm, tipicamente, tamanhos distintos. Por essa razão, quando a função `malloc()` pede um bloco com `n` bytes, o gestor do "heap"tem de procurar na lista de blocos livres um com um tamanho $\geq$ `n`. Se não existir um tal bloco, o espaço terá de ser reclamado no topo do "heap".

O gestor do "heap"consegue determinar o tamanho dos blocos libertados com a função `free()` porque cada bloco reservado no "heap"é precedido por um cabeçalho, invisível para a aplicação, que contém informação relevante sobre o bloco, tipicamente o seu tamanho em bytes e se está a ser usado ou não. No código apresentado no artigo, este cabeçalho é representado por:

```
struct header_t {
    size_t       size;
    unsigned  is_free;
};
```

Pode visualizar esta organização através da figura que se segue:



Tendo em conta esta informação, leia com atenção o artigo e estude o código apresentado para compreender como são implementadas as funções acima referidas. Compile depois o código e use-o da seguinte forma:

```
$ gcc -Wno-deprecated-declarations -o libmemalloc.so -fPIC -shared memalloc.c
$ LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
$ ls
memalloc.c     libmemalloc.so    testalloc.c
$ gcc testalloc.c -o testalloc -L. -lmemalloc
$ ./testalloc 0.5
$ ./testalloc 0.9
$ ./testalloc 0.1
...
```

Note que a definição da variável `LD_LIBRARY_PATH` faz com que o sistema operativo encontre sempre esta implementação das funções `malloc()` e `free()` em vez das implementadas na `libc` que seriam usadas por omissão.