Implementação de Linguagens
– trabalho prático –

---

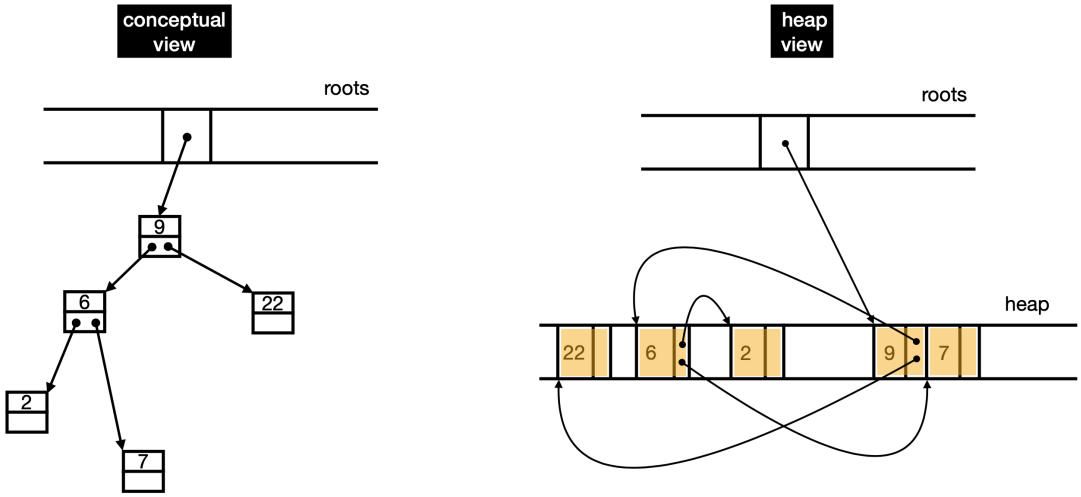
1. Considere o ficheiro `code-gc.zip` à sua disposição na página da UC no Moodle. O código em causa implementa uma aplicação simples que poderá servir de base à implementação de algoritmos de *garbage collection*. Descarregue e descomprima o ficheiro e veja o conteúdo do directório resultante:

```
$ unzip code-gc.zip
$ cd code-gc
$ ls
bistree.c
bistree.h
bool.h
collector.c
collector.h
globals.h
heap.c
heap.h
list.c
list.h
makefile
mutator.c
readme
```

Siga as instruções contidas no ficheiro `readme`.

O ficheiro `mutator.c` contém a função `main()` e executa um ciclo em que são criadas novas árvores binárias de pesquisa com um tamanho e conteúdo aleatório ou alteradas de forma aleatória árvores previamente criadas. As alterações consistem em remover nós das árvores que ficam assim disponíveis para serem recolhidos por um *garbage collector*. Em cada iteração do ciclo, o código decide de forma aleatória se adiciona ou remove nós ao heap. A decisão é baseada no único argumento passado à função `main()`, um número no intervalo $(0, 1)$.

As raízes das árvores estão guardadas numa lista `roots`. A árvore binária de pesquisa está definida e implementada em `bistree.h` e `bistree.c`. A lista está definida e implementada em `list.h` e `list.c`. Na implementação fornecida, o espaço para os nós das árvores é reservado através de uma chamada à função `my_malloc()` que é invocada no ficheiro `bistree.c`. A dita função reserva espaço no heap implementado em `heap.h` e `heap.c`.



Note que cada bloco de bytes reservado no heap é precedido por um *header* que contém informação sobre o tamanho do bloco e sobre se está ou não a ser usado, i.e., se está acessível a partir de uma das raízes em `roots`. Esta informação, invisível para o programador, é crucial para o colector.

```
typedef struct {
   unsigned int marked;
   unsigned int size;
} _block_header;
```

Associadas ao heap existem funções que implementam colectores que devem ser implementadas em `collector.c` e cujos tipos se encontram em `collector.h`. Note que na inicialização do heap uma dessas funções é lhe passada como argumento.

Note ainda o `makefile` associado ao código. Compile o programa e experimente-o:

```
$ make -f makefile all
...
$ ./mutator 0.5       /* adicionar/remover nós com probabilidade 50%-50% */
$ ./mutator 0.9       /* adicionar/remover nós com probabilidade 10%-90% */
```

2. Com base no código anterior e na descrição dos algoritmos que pode encontrar no livro recomendado (Jones, Hosking e Moss), escreva o código para os colectores: Mark&Sweep, Mark&Compact e Copy-Collection, cujos protótipos vazios podem ser vistos no ficheiro `collector.c`:

```
void mark_sweep_gc(List* roots) { ... }

void mark_compact_gc(List* roots) { ... }

void copy_collection_gc(List* roots) { ... }
```

Se possível, mantenha intacta a API fornecida. Observe o comportamento dos algoritmos com o programa `mutator` para diferentes valores da probabilidade. Note em particular o tempo de execução ou a fracção de tempo dispendida em GC.

3. Implemente um Generational Garbage Collector:

```
void generational_gc(List* roots, ....) { ... }
```

Assuma que o heap está dividido em 2 regiões: *eden* e *tenured*. O colector deve receber os seguintes parâmetros:

(a) a percentagem do heap reservada para a região *eden* - o espaço complementar será usado pelo *tenured*;

(b) o número de coleções a que um objecto tem de sobreviver para ser promovido da região *eden* para a *tenured*;

(c) os colectores para as regiões *eden* e *tenured* (pode usar o Mark&Sweep e Copy-Collection que implementou no ponto 2, respectivamente).

Pense em que condições os colectores serão invocados.

4. O código para os pontos 2 e 3 deve ser enviado para `lmlopes@fc.up.pt` incluído num ficheiro `.zip` com o código original, ao qual deve ainda acrescentar um ficheiro de texto `readme` onde descreve como testar os algoritmos. A data limite de entrega é 24 de Maio (até às 23:59). O trabalho é individual e tem de ser apresentado pelo próprio para lhe ser atribuída uma nota de acordo com os seguintes critérios:

- 2 relatórios de progresso obrigatórios em aula (30%) - quem não puder frequentar aulas pode fazer o relatório de progresso noutro horário a combinar ou via Zoom;
- código e apresentação final (70%).

Os relatórios de progresso são informais: vocês vêm à aula e explicam-me em que parte do trabalho estão, que dificuldades tiveram/estão a ter, mostram-me o código. A apresentação será feita durante os dias 28 a 30 de Maio, num horário a combinar com o docente.