IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia de Setúbal

ATAD 2019/20

Algoritmos e Tipos Abstratos de Dados

Enunciado de Laboratório
AVALIADO

5

Gestão de Memória Dinâmica

Objetivos
- Domínio e aplicação dos conceitos de gestão de memória dinâmica.

Pré-requisitos:
- Valgrind instalado de acordo com o tutorial;
- Última versão do template de submissão (no Moodle).

Notas importantes:

- Deverá existir um `makefile` que permita compilar os níveis individuais no executável `prog` e com a *flag* `-g` (símbolos de debug), e.g.:
```
n1:
    gcc -o prog -g main1.c
n2:
    gcc -o prog -g main2.c
...
```
- Deverá garantir que possui o **valgrind** instalado e incluir o seguinte *bash script* na pasta do projeto com o nome `mem_check.sh` (indicações adicionais em Anexo):
```
!#/bin/sh

valgrind --leak-check=full --show-reachable=yes --track-origins=yes ./prog
```
- **No final deste enunciado** possui o "esqueleto" inicial para o primeiro programa.
- **No template de submissão do relatório deve colocar o output da execução do *valgrind* em cada nível**.

Nível 1

1. Crie o ficheiro `main1.c` com o código fornecido que conterá todo o código deste nível.
2. Implemente a função `int* fibArrayCreate(int n)` que devolve um array alocado contendo os *n* primeiros números da sequência *fibonacci*.
3. Complete a função `main`:
   - invoque esta função e passe como argumento o número introduzido pelo utilizador;
   - imprima o endereço do array alocado pela função;
   - imprima o conteúdo do array;
   - liberte a memória alocada;
   - imprima novamente o endereço do array alocado pela função;
4. Teste o programa com o *valgring* (ver notas no final do enunciado) e garanta que não existem *memory leaks*.
   Resultado de execução esperado (endereço pode variar):

```
Length of fib sequence?: 5
Address of array = 0x55e63b1a5ac0
{1, 1, 2, 3, 5}
Address of array = 0x55e63b1a5ac0
```

5. Forneça a documentação *doxygen* para as funções `fib` e `fibArrayCreate`.
6. Inclua no relatório o resultado da execução do *valgrind*.

Nível 2

7. Crie o ficheiro `main2.c` com o código resultante de `main1.c`.
8. Crie a função `int* fibArrayCopy(int *arr, int size)` que devolve uma **cópia** de `arr` – novo array alocado com cópia dos elementos de `arr`.
9. Modifique a função `main` por forma a reproduzir o output exemplificativo seguinte e teste com o *valgrind*.
   Resultado de execução esperado (endereço pode variar):
   ```
   Address of array = 0x560dfe668ac0
   {1, 1, 2, 3, 5}
   Address of array copy = 0x560dfe668ae0
   {1, 1, 2, 3, 5}
   Address of array = 0x560dfe668ac0
   ```
10. Forneça a documentação *doxygen* para a função `fibArrayCopy`.
11. Inclua no relatório o resultado da execução do *valgrind*.

Nível 3

12. Crie o ficheiro `main3.c` contendo o conteúdo novamente de `main1.c`.
13. Crie a função `void fibArrayDestroy(int **arr)` que *recebe o endereço de um array* alocado previamente e tem como propósito libertar a memória alocada respetiva e atribuir o valor `NULL` a esse endereço passado por referência.
14. Modifique a função `main` por forma a reproduzir o output exemplificativo seguinte e teste com o *valgrind*.
    Resultado de execução esperado (endereço pode variar):
    ```
    Length of fib sequence?: 7
    Address of array = 0x55b26a7a5ac0
    {1, 1, 2, 3, 5, 8, 13}
    Address of array = (nil)
    (NULL)
    ```
    O último `(NULL)` vem da invocação de `fibArrayPrint`.
15. Forneça a documentação *doxygen* para a função `fibArrayDestroy`.
16. Inclua no relatório o resultado da execução do *valgrind*.

Nível 4

17. Crie o ficheiro `main4.c` com o conteúdo resultante do ficheiro `main3.c`.
18. Crie a função `void fibArrayExpand(int **ptArr, int *size)` cujo propósito é de "expandir" para o dobro do tamanho um array existente (recebido o seu endereço).
    - Duplica o tamanho do array, através da função `realloc` e adiciona à segunda metade a continuação da sequência *fibonacci*;
    - Altera o valor de `*size` para o dobro, por forma a manter a consistência da informação.

Exemplo parcial:

```
int *v = fibArrayCreate(n);
fibArrayPrint(v, n);
fibArrayExpand(&v, &n);
fibArrayPrint(v, n);
```

resultará em:

```
Length of fib sequence?: 4
{1, 1, 2, 3}
{1, 1, 2, 3, 5, 8, 13, 21}
```

19. Modifique a função `main` por forma a reproduzir o output exemplificativo seguinte e teste com o *valgrind*.
    Resultado de execução esperado (endereço pode variar):

```
Length of fib sequence?: 8
Address of array = 0x56394c6afac0
{1, 1, 2, 3, 5, 8, 13, 21}
{1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987}
Address of array = (nil)
(NULL)
```

20. Forneça a documentação *doxygen* para a função `fibArrayExpand`.
21. Inclua no relatório o resultado da execução do *valgrind*.

Nível 5

22. Crie o ficheiro `main5.c` com o conteúdo de `main4.c`.
23. Defina os seguintes tipos de dados no início do ficheiro:

```
typedef struct array {
    int *numbers;
    int size;
} Array;

typedef struct array* PtArray;
```

    - `struct array`: cujo propósito é o de agrupar um ponteiro para um array e o seu tamanho;
    - `PtArray`: que representa um ponteiro para uma instância de `struct array`.

24. Adapte todo o código anterior por forma a utilizar estes novos tipos de dados. Os protótipos das novas funções deverão ser os seguintes:

```
PtArray fibArrayCreate(int n);
void    fibArrayPrint(PtArray arr);
void    fibArrayDestroy(PtArray *ptArray);
void    fibArrayExpand(PtArray arr);
```

    O **output esperado** será igual ao do nível 4.

25. Forneça a documentação *doxygen* para todas as funções.
26. Inclua no relatório o resultado da execução do *valgrind*.

Anexos

Esqueleto main1.c

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

int     fib(int n);
int*    fibArrayCreate(int n);
void    fibArrayPrint(int *arr, int size);

int main() {

    int n;
    printf("Length of fib sequence?: ");
    scanf("%d", &n);

    /* ... */

    return EXIT_SUCCESS;
}

int fib(int n) {
    /* ... */
}

int* fibArrayCreate(int n) {
    /* ... */
}

void fibArrayPrint(int *arr, int size) {
    if(arr == NULL) {
        printf("(NULL)");
        return;
    }

    printf("{");
    for(int i=0; i<size-1; i++) {
        printf("%d, ", arr[i] );
    }
    printf("%d}\n", arr[size-1]);
}
```

Valgrind

Alternativas de execução (no comando executado é necessário que o executável compilado tenha o nome `prog` e que tenha sido compilado com a *flag* `-g`):

- Execute:

  ```
  $> sh mem_check.sh
  ```

- Em alternativa pode tornar o ficheiro executável:

  ```
  $> chmod +x mem_check.sh
  ```

  e daí em diante executar com:

  ```
  $> ./mem_check.sh
  ```

(fim de enunciado)