Algoritmos e Tipos Abstratos de Dados

Lab 5 | Exercícios + Template

English version

Este repositório foi criado a partir de:

- https://github.com/estsetubal-atad/CProgram_Template

Consulte o README se tiver dúvidas sobre a sua utilização.

Objetivos:

- Utilização de ponteiros;
- Passagem de argumentos por referência (vs. passagem por cópia);
- Gestão de memória dinâmica (heap).

Referências:

- *"Linguagem C"*, *e-book* disponível no Moodle.

As funções solicitadas deverão ser definidas e implementadas no módulo `fortnite`, salvo indicação em contrário.

Nível 1 (Duração estimada: < 15min)

1. Compile e execute o programa, verificando a execução do mesmo.
2. Implemente a função `bool fortniteItemBuy(const char* name, FortniteItem arr[], int arrLength)` que marca o item com o nome `name` como *comprado* (campo `owned = true`).
3. Implemente no `main` um excerto de código que solicite ao utilizador o nome do item a comprar. Imprima novamente a loja; o item, se encontrado, deverá aparecer como comprado; caso contrário, deverá apresentar uma mensagem de erro. Teste o programa.

Nível 2 (Duração estimada: ~ 15min)

4. Implemente a função `PtFortniteItem fortniteItemSearch(const char* name, FortniteItem arr[], int arrLength)` que devolve o *endereço* do item com o nome `name`; devolve `NULL` se não existir.
5. Modifique a função `fortniteItemBuy` por forma a utilizar internamente a função `fortniteItemSearch`. Verifique que o programa mantém a funcionalidade.

Nível 3 (Duração estimada: ~ 15min)

6. Implemente a função `FortniteItem* fortniteArrayCopy(FortniteItem arr[], int arrLength)` que devolve um *array alocado dinamicamente*, contendo uma cópia dos items em `arr`.
   - Note que sabe exatamente o tamanho do array a ser alocado, i.e., `arrLength`; depois basta copiar os items um a um.
7. Adicione ao `main` o código para testar esta função; crie a cópia, imprima esse array e não se esqueça de libertar a memória antes do programa terminar.
8. Utilize o **valgring** para verificar que não existem *memory leaks*. Por conveniência, este repositório inclui o *script* `mem_check.sh`.
   - No terminal, execute: `$> bash mem_check.sh`.

Nível 4 (Duração estimada: ~ 30min)

Nas duas próximas funções solicitadas não consegue saber de antemão o tamanho do array a ser alocado; deverá ir aumentando o tamanho do array, i.e., com `realloc` à medida do necessário.

9. Implemente a função `FortniteItem* fortniteFindFreeItems(FortniteItem arr[], int arrLength, int *itemSize)` que:
   - Devolve por retorno um array alocado dinamicamente com uma cópia dos items gratuitos na loja, i.e., onde o campo `vbucks == 0`.
   - Devolve por referência (`int *itemSize`)o número de items gratuitos e, consequentemente, o tamanho do array alocado/devolvido.
10. Adicione no `main` o código necessário para "comprar" automaticamente todos os items gratuitos; **o array a ser modificado é o array criado no Nível 3**. Teste o programa.
11. Implemente e teste a função `FortniteItem* fortniteFindRarityItems(FortniteItem arr[], int arrLength, const char *rarity, int *itemSize)` que:
    - Devolve por retorno um array alocado dinamicamente com uma cópia dos items cuja raridade é a solicitada ("string" `rarity` - `const` significa que não pode ser alterada dentro da função);
    - Devolve por referência (`int *itemSize`)o número de items gratuitos e, consequentemente, o tamanho do array alocado/devolvido.
    - É uma simples adaptação da função `fortniteFindFreeItems`!

Nível 5 (Duração estimada: < 30min)

12. Implemente a função `bool fortniteAddNewItem(FortniteItem item, FortniteItem *arr[], int *pArrLength)` que adiciona um novo item

`item` ao array `arr`. Note que:

- O valor passado por referência (`FortniteItem *arr[]`) contém o endereço da variável que, por sua vez, contém o endereço do array; só assim poderá ser alterado dentro da função, caso o `realloc` altere o endereço do bloco;
  - A invocação desta função será feita, e.g., `bool res = fortniteAddNewItem(<item>, &shop, &shopSize)` - note a passagem do endereço do array (`&shop`).
- O valor passado por referência (`int *pArrLength`) contém o tamanho atual da loja e será incrementado se o item for adicionado com sucesso; terá de aumentar dinamicamente o array para esse efeito!
- Devolve `true` se o item for adicionado com sucesso, `false` se não for possível alocar mais memória.
- Não pode existir nenhum item com o mesmo nome; devolva `false` caso essa situação ocorra.

13. Teste a função anterior e corra o **valgrind**.
    - **Consegue invocar esta função sobre o array `shop` (na memória stack)?**
    - **E sobre a cópia da loja (array alocado dinamicamente)?**
    - Entende o porquê? A função `realloc` só pode ser executada sobre blocos alocados com `malloc/calloc`.

`@bruno.silva`
`(EOF)`