Instituto Superior de Engenharia de Lisboa

Programação II

2020/21 – 2.º semestre letivo

Teste de época normal

2021.01.29

1. [4 valores] Considere as definições das funções `f1`, `f2`, e `f3`. Na linha 1 do troço de código seguinte deverá inserir o algarismo das unidades do seu número de aluno no local indicado. Por exemplo: se o seu número de aluno fosse o 12345, aquela linha ficaria assim: `#define ALG 5`

   Nota importante: a utilização do algarismo errado penalizará significativamente a avaliação da alínea b) deste grupo.

```
1   #define ALG algarismo_das_unidades_do_seu_número_de_aluno
2   #include <stdio.h>
3   #include <stdlib.h>
4   #include <string.h>
5   int f2(int * e) {
6       return *e == ALG;
7   }
8   void f3(int * e) {
9       *e = -ALG;
10  }
11  int * f1(int a[], size_t *size, int (*fin)(int * e), void (*fac)(int * e)) {
12      size_t i = *size; int *b ;
13      while(i--)
14          if((*fin)(&a[i])){
15              if((b = realloc(a, (*size + 1) * sizeof(int))) == NULL) return b;
16              memmove(&b[(*size)++], &b[i], sizeof(int));
17              (*fac)(&b[i]);
18          }
19      return b;
20  }
```

   Na caixa seguinte consta um extrato do *output* do comando `man memmove` executado em consola.

```
(…)
  void *memmove(void *dest, const void *src, size_t n);


DESCRIPTION
  The memmove() function copies n bytes from memory area src to memory area dest. The
memory areas may overlap: copying takes place as though the bytes in src are first
copied into a temporary array that does not overlap src or dest, and the bytes are
then copied from the temporary array to dest.
(…)
```

   No troço de programa seguinte consta um exemplo de utilização das funções **`f1`**, **`f2`**, e **`f3`**.

```
...   int *a;
...   (…)  // Preenchimento do array dinâmico a.
30    size_t i, size=24;
31    int *b = f1(a, &size, f2, f3);
32    for(i=0; i<size; i++)
33        printf("%d ", b[i]);
```

a) [1] Comente a função **f1** e apresente, correta e completamente preenchido, o respetivo cabeçalho descritivo seguindo o formato do que se apresenta na caixa seguinte, que está preenchido (como exemplo) para a função **memmove**.

```
/*-----------------------------------------------------------------------------

Nome da função: memmove


Descrição: Esta função copia n bytes da memória apontada por src para a memória apontada
por dest. As áreas de memória podem sobrepor-se: a cópia ocorre como se os bytes a copiar
fossem primeiro copiados para um array temporário, não sobreposto às áreas de memória
apontadas por src e dst, e fossem depois copiados para a memória apontada por dest.


Parâmetros:

void *dest: referência para a memória para onde é realizada a cópia.

const void *src: referência para a memória de onde é realizada a cópia.

size_t n: número de bytes a copiar.


Retorno:

void: não tem tipo de retorno.

-----------------------------------------------------------------------------*/
```

b) [1] Apresente, justificando, os valores produzidos em *standard output* resultantes da execução do troço de código anterior, admitindo que (imediatamente antes da execução da linha 31) o conteúdo do *array* **a** é o seguinte:

2	-3	1	3	0	4	7	8	6	-5	7	1	8	9	5	-9	5	3	0	6	2	-1	9	4

Nota: a ausência de justificação dos valores produzidos em *standard output* penalizará significativamente a avaliação desta alínea.

c) [2] Considere que se pretende agora realizar uma função **f1a** com a mesma funcionalidade da função **f1** mas com maior generalidade, que possa operar sobre *arrays* de outros tipos e não apenas *arrays* de elementos do tipo **int**. Sendo uma versão da função **f1,** a função **f1a** deve seguir **de muito perto** o algoritmo da função **f1**.

Escreva a definição da função **f1a**, adicionando ou removendo parâmetros se necessário. Escreva a definição da função com uma indentação correta, usando obrigatoriamente comentários.

Rescreva a linha onde é usada a função **f1**, no exemplo de utilização anterior, substituindo-a pela utilização da função **f1a**, igualmente aplicada ao *array* **a**.

Nota: desaconselha-se veementemente a escrita da definição desta função sem a utilização de comentários porque penalizará significativamente a avaliação desta alínea.

2. [7 valores] Pretende-se construir uma estrutura de dados para representar as diretorias de um sistema de ficheiros, registando os caminhos existentes e os ficheiros contidos em cada localização.

   Admita que, para obter a informação do sistema de ficheiros, dispõe das funções

   void dirStart(char *initialPath);

   que inicia a pesquisa das diretorias no caminho indicado pelo parâmetro; deve ser invocada antes de iniciar o uso da seguinte,

   int dirNext(char *path, char *name, int *size);

   que obtém os dados de um ficheiro. Em **path** e **name** deve passar dois *arrays* onde a função depositará, respetivamente, o caminho e o nome de um ficheiro; em **size** deve passar o endereço de uma variável que será afetada com a dimensão do ficheiro. Esta função, chamada repetitivamente, vai percorrendo todas as diretorias dependentes do caminho inicial e em cada retorno indica um dos ficheiros que encontra. Pode haver várias ocorrências que indicam o mesmo caminho com nome de ficheiro diferente (são os vários ficheiros de cada diretoria).

   Em caso de sucesso a função retorna 1; quando não houver mais ficheiros, retorna 0 sem afetar as variáveis dos parâmetros. Considere que as dimensões máximas de caminho e nome são ambas de 100 carateres.

   Propõe-se os tipos seguintes para representar a informação dos ficheiros

```
typedef struct{                 // Descritor de um ficheiro
    char *name;                 // nome do ficheiro, string em espaço alojado dinamicamente
    int size;                   // dimensão do ficheiro
    struct pathList *path;      // aponta o caminho da localização do ficheiro
} FileEntry;

typedef struct{                 // Descritor de um conjunto de ficheiros
    FileEntry **entrys;         // aponta array de ponteiros, alojado dinamicamente
    int count;                  // quantidade de elementos do array entrys
} FileSet;

typedef struct pathList{        // Nó de lista de localizações das diretorias
    struct pathList *next;      // ponteiro de ligação em lista
    char *path;                 // caminho da localização, string alojada dinamicamente
    FileSet files;              // ficheiros existentes nesta localização
} PathList;
```

   Para desenvolver as funções seguintes, sempre que achar conveniente pode escrever funções auxiliares, bem como utilizar funções desenvolvidas noutros exercícios do teste.

   a) [1,5] Escreva a função

      void fileSetAdd(FileSet *ns, FileEntry *fe, PathList *p);

      que adiciona ao conjunto **ns** o ficheiro **fe**, registando também o caminho **p** da sua localização. Deve usar a função **realloc** de biblioteca, adicionando um elemento de cada vez ao *array*.

   b) [1,5] Escreva a função

      void fileSetSort(FileSet *ns);

      que ordena o conjunto de ficheiros os seguintes critérios: 1.º ordem alfabética crescente do nome; 2.º ordem alfabética crescente do caminho; 3.º dimensão.

   c) [2] Escreva a função

      PathList *pathListBuild(char *initialPath);

      que constrói a lista de localizações, registando os respetivos ficheiros, com a informação obtida das chamadas sucessivas à função **dirNext**. A lista tem ordem alfabética crescente dos caminhos representados.

   d) [2] Escreva a função

      void pathListOrganize(PathList *list);

      que percorre a lista de localizações e ordena, em cada localização, o conjunto de ficheiros, usando a função **fileSetSort**.

3. [6 valores] No grupo anterior foram definidos os tipos seguintes e especificada as funções que os utilizam

```
typedef struct{                 // Descritor de um ficheiro
        char *name;             // nome do ficheiro, string em espaço alojado dinamicamente
        int size;               // dimensão do ficheiro
        struct pathList *path;  // aponta o caminho da localização do ficheiro
} FileEntry;

typedef struct{                 // Descritor de um conjunto de ficheiros
        FileEntry **entrys;     // aponta array de ponteiros, alojado dinamicamente
        int count;              // quantidade de elementos do array entrys
} FileSet;

typedef struct pathList{        // Nó de lista de localizações das diretorias
        struct pathList *next;  // ponteiro de ligação em lista
        char *path;             // caminho da localização, string alojada dinamicamente
        FileSet files;          // ficheiros existentes nesta localização
} PathList;
```

```
void fileSetAdd( FileSet *ns, FileEntry *fe, PathList *p );

void fileSetSort( FileSet *ns );

PathList *pathListBuild( char *initialPath );

void pathListOrganize( PathList *list);
```

Pretende-se agora criar uma estrutura de dados para permitir a pesquisa eficiente de ficheiros por nome, permitindo obter as respetivas localizações. A estrutura é baseada numa árvore binária de pesquisa. Cada nó, com o tipo **BstNode**, armazena um conjunto de ficheiros com o mesmo nome que estão em diversas localizações, referenciadas nos respetivos descritores de ficheiro.

```
typedef struct bstNode{             // Descritor de nó da árvore
        struct fileTree *left, *right;  // ponteiros de ligação na árvore
        FileSet files;              // conjunto de ficheiros
} BstNode;
```

Para desenvolver as funções seguintes, sempre que achar conveniente pode escrever funções auxiliares, bem como utilizar funções desenvolvidas noutros exercícios do teste.

a) [2] Escreva a função

```
BstNode *bstAdd( BstNode *r, FileEntry *fe );
```

que adiciona à árvore, identificada pela raiz **r**, o ficheiro **fe** à árvore, colocando-o no nó que tem ficheiros com o mesmo nome, se já existir, ou adicionando um nó novo. Os nós novos são inseridos como folhas. A ordenação da árvore é alfabética pelo nome dos ficheiros existentes nos nós; os menores são indicados pela ligação **left**. A função retorna a raiz da árvore atualizada.

b) [2] Escreva a função

```
BstNode *bstBuild( PathList *list );
```

que constrói uma árvore a partir dos dados contidos na lista de localizações **list**. A função retorna a raiz da árvore construída.

c) [2] Escreva a função

```
void bstSelectPrint( BstNode *r, char *name );
```

que seleciona, na árvore com raiz **r**, o nó com ficheiros identificados pelo nome, indicado por **name**, e apresenta através de *standard output* as localizações em que eles se encontram.

4. [3 valores] Considere um conjunto de módulos escritos em linguagem C, compilados individualmente com o comando "**gcc -c *.c**". Na caixa abaixo apresenta-se, dos módulos compilados, as respetivas listas de símbolos, resultantes do comando "**nm *.o**", classificados com as abreviaturas: `T`, *public text*; `t`, *private text*; `U`, *undefined*; `d`, *private BSS data*.

```
copy.o:
0000000000000000    T copy

data.o:
0000000000000015    T dAdd
000000000000004d    T dEnd
000000000000003f    T dGetIdx
0000000000000034    T dSize
0000000000000000    T dStart
                    U fopen
                    U malloc
                    U realloc
norm.o:
0000000000000000    T normalize

print.o:
0000000000000000    T print
                    U printf

read.o:
0000000000000000 b f
                    U fclose
                    U fgets
                    U fopen
                    U malloc
                    U normalize
0000000000000087    T rEnd
0000000000000048    T rItem
0000000000000000    T rStart
0000000000000026    t split
                    U strtok
```

Considere também que há um módulo de aplicação, **demo.c**, contendo a função **main** seguinte

```
int main( int arc, char **argv ){
        rStart( argv[1] );
        Item * d;
        while( ( d = rItem() )!= NULL ){
                dAdd( d );
        }
        rEnd();
        for( int i = 0; i < dSize(); ++i ){
                print( dGetIdx( i ) );
        }
        return 0;
}
```

Admita que são criados e usados no processo de compilação os *header files* **data.h**, **norm.h**, **print.h** e **read.h**, cada um contendo as assinaturas das funções públicas do módulo fonte (**.c**) com o respetivo prefixo. Existe também o *header file* **item.h**, com a definição do tipo **Item**, e é incluído em todos os módulos fonte.

a) [0,5] Apresente a lista de símbolos produzida por "**nm demo.c**" e a respetiva classificação (t, T, U, etc.).

b) [0,5] Diga, justificando, se há funções definidas com o atributo **static**.

c) [1] Escreva o *header file* **read.h**, tendo em conta o controlo de inclusão múltipla e considerando assinaturas das funções compatíveis com a utilização na função **main**. Indique, justificando, quais os módulos de código fonte (**.c**) onde o *header file* **read.h** deve ser incluído.

d) [1] Com base nos símbolos listados, identifique os módulos que devem ser ligados ao módulo de aplicação **demo.o** para gerar o executável. Escreva um *makefile* que produza, de forma eficiente, o executável com o nome "**demo**" a partir dos módulos fonte (**.c**).