Instituto Superior de Engenharia de Lisboa

Programação II

2023/24 – 1.º semestre letivo

Exame de Época Normal

2024.01.08

Esta prova é formada por duas partes, I e II. A **Parte I** corresponde à repetição do **1.º Teste Parcial**; a **Parte II** corresponde ao **2.º Teste Parcial**.

Todos os alunos recebem a prova completa. A realização do **2.º teste** tem a duração de **1 hora e 15 minutos**; se o aluno permanecer após esse tempo, realiza **exame completo**, com a duração de **2 horas e 30 minutos**.

Cada parte é cotada para 20 valores; no caso de exame completo, a classificação é a média aritmética das duas partes.

Nota importante: Valoriza-se a escrita de código que inclua comentários esclarecedores da implementação seguida e que contenha indentação legível.

Parte I – Repetição do 1.º Teste Parcial

(Se realiza apenas o 2.º Teste Parcial, deve ignorar esta parte e passar de imediato à Parte II)

1. [4 valores]

Escreva a função

```
void activateBits(long long *value, int step);
```

que modifica o valor da variável indicada por `value`, ativando (com o valor 1) os bits localizados nas posições múltiplas de `step`, ou seja, na posição 0, `step`, 2*`step`, 3*`step`, …

Por exemplo:

Colocar todos os bits a 1	`activate_bits(&myVar, 1);`
Colocar bits a 1 nas posições pares	`activate_bits(&myVar, 2);`
Colocar bits a 1 nas posições múltiplas de 3 (0, 3, 6, 9, ...)	`activate_bits(&myVar, 3);`

2. [8 valores]

Para a realização da **Série de Exercícios 2** foi utilizado o tipo `Store` como descritor de um *array* (com alojamento estático) de estruturas do tipo `TagData`, sendo estas estruturas utilizadas para representação e armazenamento das *tags* existentes em ficheiros MP3. Uma *tag* representa uma faixa (*track*) de um álbum de música.

```
typedef struct{
  char filename[MAX_FILENAME + 1];
  char title[MAX_TITLE + 1];
  char artist[MAX_ARTIST + 1];
  char album[MAX_ALBUM + 1];
  short year;
  char comment[MAX_COMMENT + 1];
  char track;
  char genre;
} TagData;
```

Admita que as constantes simbólicas indicadas em maiúsculas contêm valores adequados à representação pretendida.

No troço de código seguinte é utilizado o tipo `StoreArr` com objetivo semelhante ao do tipo `Store` mas definido com mais um membro, o membro `comp`.

```
typedef struct{
  TagData *data;  // aponta para array de tags (o array tem alojamento estático)
  int space;      // capacidade do array; quantidade de elementos
  int count;      // quantidade de elementos preenchidos
  int (*comp)(const void *, const void *); // Função de comparação para
                                           // ordenar o array data.
} StoreArr;
```

Considere o troço de código que consta na caixa seguinte:

```
1   (…)
2   int trackComp(const void *e1, const void *e2){
3     /* … */
4   }
5   #define TAGDATA_ARRAY_SIZE 100
6   TagData a[TAGDATA_ARRAY_SIZE];
7   int main(){
8     StoreArr sa;
9     setupStoreArr(&sa, a, TAGDATA_ARRAY_SIZE, trackComp);
10    /* Preenchimento de sa...*/
11    sortStoreArr(&sa);
12    TagData newTag;
13    /* Preenchimento de newTag...*/
14    if(insertOrd(&sa, &newTag))
15      printf("Insercao bem sucedida\n");
16    return 0;
17  }
```

a) [2] Escreva a função

```
void setupStoreArr(StoreArr *a, TagData *data, int nElem,
                   int(*comp)(const void*, const void*) );
```

que configura um descritor de um *array* do tipo `StoreArr`, no estado vazio mas com alojamento (estático) já realizado para `nElem` elementos, registando internamente a função indicada por `comp`, associada para utilização futura, como se exemplifica na linha 9 da caixa com troço de código.

b) [2] Escreva a função

```
void sortStoreArr(StoreArr *arr);
```

que ordena o *array* pertencente ao descritor indicado por `arr` com o critério implementado pela função de comparação associada, registada no campo `comp`. Tem de usar a função `qsort` da biblioteca normalizada.

```
void qsort(void *base, size_t nitems, size_t size,
           int (*compare)(const void *, const void*));
```

c) [2] Escreva a função de comparação de nome `trackComp`, destinada a ser usada pelas funções das alíneas anteriores, que origine a ordenação do *array* de *tags* pertencente a um descritor do tipo `StoreArr` por ordem crescente do membro `track` das *tags*. Considere a definição incompleta da função como apresentada na linha 2 da caixa com troço de código.

d) [2] Escreva a função

```
int insertOrd(StoreArr *arr, TagData *tag);
```

que insere no *array* de *tags* pertencente ao descritor indicado por `arr` uma nova *tag*, indicada pelo parâmetro `tag`. A função admite que o *array* está ordenado pelo critério da respetiva função

`comp` e tem de mantê-lo ordenado pelo mesmo critério, devendo retornar 1 se a inserção for bem sucedida e 0 se não for bem sucedida (por não haver espaço disponível no *array*), caso em que não deverá alterar nada no *array* pertencente ao descritor indicado por `arr`. A função não pode usar funções auxiliares de ordenação, como a função `qsort` da biblioteca normalizada.

3. [4 valores]

Pretende-se produzir mnemónicas de nomes contidos em *strings*, com várias palavras separadas por um ou vários espaços. A mnemónica contém a primeira palavra, exatamente como está escrita, seguida de um espaço e de uma sigla formada pelas iniciais das palavras seguintes, em maiúsculas.

Exemplos: `"Luis Vaz de Camoes "` → mnemónica: `"Luis VDC"`;
`" RIta REIS dos SANtos"` → mnemónica: `"RIta RDS"`.

Escreva a função

```
void mnemonic( char *str );
```

que modifica a *string* indicada por `str`, depositando, no mesmo espaço, a mnemónica resultante do seu conteúdo inicial.

Propõe-se que use a primitiva «`int toupper(int c);`» declarada no *header file* `ctype.h`.

4. [4 valores]

Considere um conjunto de módulos fonte, em linguagem C, e a utilização do comando "`gcc -c …`" para produzir os módulos compilados. A caixa ao lado contém os respetivos símbolos, apresentados pela ferramenta "`nm`". Considere também o módulo de aplicação "`aplic.c`" com o código fonte reproduzido abaixo.

Admita que são criados e usados no processo de compilação os *header files* `width.h`, `height.h`, `area.h`, `ratio.h` e `scale.h`, cada um contendo as assinaturas das funções públicas do respetivo módulo fonte (`.c`).

Além destes, existe ainda o *header file* `rectangle.h` com a definição do tipo `Rectangle`, o qual é incluído em todos os *header files* anteriormente indicados.

```
area.o:
0000000000000000 T area
                 U height
                 U width

height.o:
0000000000000000 T height

ratio.o:
                 U height
0000000000000000 T ratio
                 U width

scale.o:
0000000000000000 t multiply
0000000000000029 T scale

width.o:
0000000000000000 T width
```

aplic.c

```
int main(){
    Rectangle r;
    scanf( "%lf%lf", &r.width, &r.height );
    printf( "Rectange ratio: %lf\n", ratio( &r ) );
    scale( &r, 2.0 );
    printf( "Scaled: %lf x %lf; ratio: %lf\n",
            r.width, r.height, ratio( &r ) );
    return 0;
}
```

a) [2] Escreva o *header file* `scale.h`, tendo em conta o controlo de inclusão múltipla e supondo a assinatura da função compatível com a utilização na função `main`. Indique, justificando, quais os módulos de código fonte (`.c`) onde o *header file* `scale.h` deve ser incluído. Descreva o motivo para não colocar a assinatura da função `multiply`.

b) [2] Tendo em conta as dependências existentes, apresente a lista dos módulos compilados que devem ser ligados ao módulo de aplicação (`aplic.o`) para gerar o executável. Escreva um *makefile* que produza, de forma eficiente, o executável com o nome "`aplic`" a partir dos módulos fonte (`.c`) estritamente necessários.

Parte II – 2.º Teste Parcial

Em todos os exercícios desta parte, por simplificação, assuma que o **alojamento dinâmico** é **sempre bem sucedido**, não ocorrendo falta de memória de *heap*.

5. [9 valores]

Pretende-se armazenar, em alojamento dinâmico, um conjunto de eventos. Cada evento é obtido de uma linha de texto iniciada pela descrição, podendo ter várias palavras, seguida de ':' e da data, em sequência de ano, mês e dia. Os campos da data são separados por um ou mais carateres de vários possíveis, que são '/', '-', '.' ou espaço. Para ser válida a *string* deve estar completa, com todos os campos especificados.

Exemplos: "Pg2 - 1.º teste parcial: 2023-11-10"
"ISEL, Fim das aulas: 2023/12.20"
"Pg2 - Exame de epoca normal: 2024 01--08"

Os eventos são registados em elementos, alojados dinamicamente, com o tipo seguinte:

```
typedef struct{
    char *desc;   // descrição - string alojada dinamicamente
    int  year;    // ano
    short month;  // mês
    short day;    // dia
}Event;
```

a) [3] Escreva a função

```
char *eventSplit(char *text, int *yPtr, int *mPtr, int *dPtr);
```

destinada a separar e identificar os dados de um evento, contidos na *string* indicada por `text`, criando uma réplica da descrição, em alojamento dinâmico, e afetando as variáveis indicadas por `yPtr`, `mPtr` e `dPtr`, respetivamente, com os valores de ano, mês e dia. Em caso de sucesso, retorna a réplica da *string* de descrição criada; se ocorrer insucesso, devido a conteúdo de `text` incompleto, retorna NULL. Neste caso, deve terminar sem deixar espaço inadequadamente alojado.

Por simplificação, não se pretende que verifique se os valores da data pertencem às respetivas gamas de validade.

Propõe-se que use as funções `strtok` e `atoi` da biblioteca normalizada.

```
char *strtok(char *string, char *delimiters);
int atoi(char *string);
```

b) [3] Escreva a função

```
Event *eventCreate(char *text);
```

destinada a criar, em alojamento dinâmico, um elemento `Event` preenchido com os dados obtidos da *string* `text`. A função retorna o endereço do elemento criado ou NULL, em caso de insucesso. Neste caso, a função não deve deixar espaço inadequadamente alojado. Deve utilizar a função `eventSplit`.

c) [3] Com o propósito de adicionar anotações à descrição de um evento, escreva a função

```
void eventDescAppend(Event *event, char *str);
```

que modifica o campo `desc` da *struct* indicada por `event`, adicionando, no final da *string* existente, o texto indicado por `str`. Deve usar a função de biblioteca adequada para assegurar o espaço necessário ao novo conteúdo da *string*.

6. [5 valores]

Pretende-se organizar, numa lista simplesmente ligada, a criação e acesso de um conjunto de eventos com o tipo `Event`, definido no anteriormente. A lista é ordenada crescentemente pela data dos eventos. Considere o nó de lista representado pelo tipo seguinte.

```
typedef struct listEvPtr{
  struct listEvPtr *next;
  Event *ptr;
} ListEvPtr;
```

a) [2] Escreva a função

```
int compareEventDate(Event *event1, Event *event2);
```

que compara as datas de dois eventos, retornando um valor negativo, zero ou positivo se, respetivamente, a data de `event1` for anterior, igual ou posterior à de `event2`.

b) [3] Escreva a função

```
int listInsert(ListEvPtr **headAddr, char *text);
```

destinada a criar um novo elemento com os dados obtidos de `text` e inseri-lo, ordenado, na lista cujo ponteiro cabeça é indicado por `headAddr`. Em caso de sucesso, retorna 1; se falhar a criação do evento, não deve modificar a lista e retorna 0. Deve usar as funções `compareEventDate` e `eventCreate`.

7. [6 valores]

Pretende-se suportar, através de uma árvore binária de pesquisa, o acesso a elementos previamente criados com o tipo `Event`, definido anteriormente. A árvore é ordenada pelas *string* de descrição dos elementos referenciados, alfabeticamente crescente de `left` para `right`.

Considere o nó de árvore binária representado pelo tipo seguinte.

```
typedef struct bstEvPtr{
  struct bstEvPtr *left, *right;
  Event *ptr;
} BstEvPtr;
```

a) [3] Escreva a função

```
int bstInsert(BstEvPtr **rootAddr, Event *event);
```

que insere na árvore binária um novo nó associado ao elemento indicado por `event`, que já existe no instante de inserção, previamente alojado e preenchido. Em caso de sucesso, retorna 1; se já existir um evento com a descrição igual, não deve modificar a árvore e retorna 0.

b) [3] Escreva a função

```
int bstPrintEvent(BstEvPtr *root, char *desc);
```

destinada a procurar na árvore indicada por `root`, a descrição indicada por `desc` e apresentar a respetiva data, em *standard output*. Em caso de sucesso, retorna 1; Se a *string* pesquisada não existir, não apresenta nada e retorna 0.