Instituto Superior de Engenharia de Lisboa

Programação II

2023/24 – 1.º semestre letivo

1.º Teste Parcial

2023.11.10

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

1. [4 valores]

Pretende-se contar os bits de menor peso a 0 numa palavra em binário natural.

Escreva a função

```
int countTrailingZeros( unsigned long long data );
```

que conta e retorna a quantidade de bits consecutivos com o valor 0 na extremidade de menor peso do parâmetro `data`. Com vista à portabilidade do código, se necessitar de identificar a dimensão da palavra deve usar o operador `sizeof` e a macro `CHAR_BIT` definida em `limits.h`.

2. [8 valores]

Para a realização da **Série de Exercícios 2** foi utilizado o tipo `TagData` para armazenamento das *tags* existentes num ficheiro MP3. Uma *tag* representa uma faixa (*track*) de um álbum de música. Admita que as constantes simbólicas indicadas em maiúsculas contêm valores adequados à representação pretendida.

```
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;
```

Considere o troço de código que consta na caixa seguinte:

```
1   (…)
2   int printTagYear(TagData *tag, void *notUsed){
3     printf("Year: %d\n", (int)tag->year);
4     return 1;
5   }
6   int main(){
7     (…) // Preenchimento do array a
8     size_t nElem = sizeof(a) / sizeof(a[0]);
9     size_t s1 = tagScan(a, nElem, printTagYear, NULL);
10
11    short date = 2002;
12    size_t s2 = tagScan(a, nElem, printRecentAlbum, &date);
13    (…)
14    return 0;
15  }
```

a) [4] Escreva a função

```
size_t tagScan( TagData a[], size_t nElem,
                int (*action)(TagData *data, void *context ),
                void * context );
```

que percorre os `nElem` elementos do *array* `a`, chamando a função `action` para cada elemento, passando-lhe um ponteiro para esse elemento no parâmetro `data` e passando-lhe ainda o parâmetro `context`. A função `tagScan` retorna a soma dos valores retornados nas sucessivas chamadas à função `action`.

Quando utilizada como consta na linha 9, a função `tagScan` produzirá em *standard output* a afixação do campo `year` de todas as *tags* que existem no *array* **`a`**, de acordo com o código da função `printTagYear` e deixará a variável **`s1`**, afetada na linha 9, com o número de elementos do *array* **`a`**.

b) [2] Escreva a função

```
int printRecentAlbum(TagData *tag, void *context);
```

que, quando usada como consta na linha 12, produzirá em *standard output* a afixação do campo `title` de todas as *tags* que existem no *array* **`a`** e cujo ano é superior ao valor contido na variável `date`, iniciada na linha 11. Admita que todas as *tags* têm no campo `title` uma *string* válida, devidamente terminada.

A função deve retornar um valor tal que a variável **`s2`**, afetada na linha 12, fique com o valor da contagem das *tags* cujo ano é superior ao valor contido na variável **`date`**.

c) [2] Utilizando a função `bsearch` da biblioteca normalizada, escreva a função

```
int yearExist (TagData *a, size_t nElem, short year);
```

que verifica se alguma *tag* existente no *array* referenciado pelo parâmetro `a` tem o ano igual ao parâmetro `year`. Se existir pelo menos uma *tag* com aquele ano, a função retorna 1; em caso contrário retorna 0. A função `yearExist` tem de utilizar a função `bsearch` da biblioteca para realizar a procura no *array* e deve admitir que este está ordenado ascendentemente por ano. Deve escrever também a função de comparação a passar por parâmetro à função `bsearch`.

Resumo da descrição da função `bsearch`:

```
void *bsearch( const void *key, const void *base,
               size_t nmemb, size_t size,
               int (*compar)( const void *key, const void *elem ) );
```

A função `bsearch` procura no *array* ordenado indicado por `base`, com `nmemb` elementos de dimensão `size` *bytes*, um elemento identificado como idêntico pela função `compar`. Esta deve comparar os dados indicados por `key` com os indicados por `elem` e retornar negativo, zero ou superior se, respetivamente, `key` é menor, idêntico ou maior.

A função `bsearch` retorna o endereço do elemento encontrado ou NULL se não existir.

3. [4 valores]

Pretende-se uniformizar a representação de matrículas de viaturas, produzindo um formato com a sequência de algarismos e letras, em maiúsculas, sem qualquer separador.

Escreva a função

```
char *uniformize( char *str );
```

que modifica a *string* indicada por `str`, de modo a uniformizar a matrícula que esta contém. Devem ser suprimidos quaisquer carateres que não sejam letras ou algarismos; na representação final, as letras devem estar em maiúsculas. Retorna a própria *string* `str`.

Propõe-se que use a primitiva `toupper` definida no *header file* normalizado `ctype.h`.

4. [4 valores]

Considere a definição de uma *struct* e um conjunto de funções, escritas como indicado na tabela:

Peça e ficheiro	**Conteúdo relevante**

Definição em `rectangle.h`	`typedef struct{`
	`  double width;`
	`  double height;`
	`} Rectangle;`

Código em `width.c`	`double width( Rectangle *rp ){`
	`  return rp->width;`
	`}`

Código em `height.c`	`double height( Rectangle *rp ){`
	`  return rp->height;`
	`}`

Código em `area.c`	`double area( Rectangle *rp ){`
	`  return width( rp ) * height (rp );`
	`}`

Código em `ratio.c`	`double ratio( Rectangle *rp ){`
	`  return width( rp ) / height (rp );`
	`}`

Código em `main.c`	`int main(){`
	`  Rectangle r;`
	`  scanf( "%lf%lf", &r.width, &r.height );`
	`  printf( "Rectange area: %lf\n", area( &r ) );`
	`  return 0;`
	`}`

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

a) [2] Escreva o *header file* `height.h`, tendo em conta o controlo de inclusão múltipla. Indique, justificando, quais os módulos de código fonte (`.c`) onde o *header file* `height.h` deve ser incluído.

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