Instituto Superior de Engenharia de Lisboa

Programação II

2020/21 – 2.º semestre letivo

Teste de época normal

2021.07.09

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 *= -1;
10  }
11  int * f1(int a[], size_t *size, void(*act)(int * e), int(*tst)(int * e) ) {
12    int *c, *b = NULL; size_t i = *size;
13    for(*size = 0; i--; )
14      if((*tst)(&a[i])){
15        if((c = realloc(b, (*size + 1) * sizeof(int))) == NULL) {
16          free(b);
17          return NULL;
18        }
19        b = c;
20        b[*size] = a[i];
21        (*act)(&b[*size]);
22        (*size)++;
23      }
24    return b;
25  }
```

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

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

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

2. [4 valores] Admita que existe o registo de membros de uma coletividade, armazenado num *array* cujos elementos são do tipo `Member` seguinte.

```
typedef struct{
    int year;
    short month;
    short day;
} Date;

typedef struct{
    int number;     // número de identificação
    char *name;     // nome, string alojada dimamicamente
    char *address;  // morada, string alojada dimamicamente
    Date birth;     // data de nascimento
} Member;
```

Pretende-se dispor de um acesso ordenado sem modificar o *array* original. Para isso é criado, em alojamento dinâmico, um *array* de ponteiros, os quais são apontados para os elementos do *array* original e em seguida ordenados.

a) [1] Escreva a função

```
int dateCompare( Date *d1, Date *d2 );
```

que, comparando o ano, mês e dia nos campos `year`, `month` e `day` de duas datas, retorna um valor negativo, zero ou superior, respetivamente, se a data indicada por `d1` é mais antiga, a mesma ou mais recente face à indicada por `d2`.

b) [1] Escreva a função

```
int birthCompare( void *e1, void *e2);
```

adequada para usar como parâmetro da função `qsort`, de modo a ordenar um `array` de ponteiros por data crescente no campo `birth` dos elementos `Member` apontados. Deve utilizar a função `dateCompare`.

c) [2] Escreva a função

```
Member **refCreateSort( Member *a, int n );
```

que aloja dinamicamente, preenche, ordena e retorna um *array* de ponteiros para os elementos do *array* indicado pelo parêmetro `a`. O parâmetro `n` é a quantidade de elementos do *array*. A ordenação é por data crescente no campo `birth` dos elementos `Member` apontados. Para realizar a ordenação deve usar a função `qsort`, passando a função de comparação `birthCompare`.

```
void qsort( void *base, size_t num, size_t size,
            int (*compar)( const void*, const void* ) );
```

3. [5 valores] Pretende-se construir um programa para gestão dos pedidos à cozinha de um restaurante. Os pedidos são organizados em lista ligada, com nós do tipo `QNode`. As listas são exploradas como filas por ordem de chegada, sob o controlo de descritores do tipo `Queue`.

```
typedef struct qNode{
    struct qNode *next; // ligação em lista
    int ident;  // número, identificador do pedido
    char *text; // descrição do pedido
} QNode;

typedef struct{
    QNode *oldest; // aponta elemento mais antigo na fila
    QNode *newest; // aponta elemento mais recente na fila
} Queue;
```

A função `qPut` seguinte adiciona um novo elemento à fila de pedidos.

```
void qPut( Queue *q, int i, char *t ){
    QNode *n = malloc( sizeof *n );
    n->ident = i;
    n->text = strdup( t );
    n->next = NULL;
    if( q->oldest == NULL )
        q->oldest = n;
    else
        q->newest->next = n;
    q->newest = n;
 }
```

a) [1] Escreva a função

```
QNode *qGet( Queue *q );
```

que retira e retorna o elemento mais antigo presente na fila ou NULL se a fila estiver vazia. Ao retirar um elemento, no caso de a fila ficar vazia os ponteiros devem ficar em condições de voltar a colocar novos elementos.

b) [2] Escreva a função

```
void qDel( Queue *q, int i );
```

que elimina o elemento identificado pelo número passado no parâmetro `i`. Deve libertar a memória alojada dinamicamente em posse do elemento eliminado.

c) [2] Escreva a função

```
void qPrt( Queue *q );
```

que apresenta, em *standard output*, o identificador e o texto descritivo de todos os pedidos em fila, do mais antigo para o mais recente.

4. [4 valores] Pretende-se construir um programa para contagem de ocorrências de palavras em textos. A estrutura proposta é uma árvore binária de pesquisa, com nós do tipo `TNode`, construída pela função `tAdd`, sucessivamente invocada por cada palavra lida do texto.

```
typedef struct {
    char *str;  // palavra representada
    int occur;  // número de ocorrências contabilizado
} Item;

typedef struct tNode{
    struct tNode *left, *right; // ponteiros de ligação na árvore
    Item data;  // palavra e respetivas ocorrências
} TNode;

void tAdd( TNode **rp, char *s ){
    if( *rp == NULL ){
        TNode *n = malloc( sizeof *n );
        n->data.str = strdup( s );
        n->data.occur = 1;
        n->left = n->right = NULL;
        *rp = n;
        return;
    }
    int c = strcmp( s, (*rp)->data.str );
    if( c == 0 )
        ++(*rp)->data.occur;
    else
        tAdd( c < 0 ? &(*rp)->left : &(*rp)->right, s );
}
```

a) [2] Escreva a função

```
int tSearch( TNode *r, char *s );
```

que retorna o número de ocorrências da palavra indicada por `s`, por pesquisa na árvore identificada pela raiz `r`.

b) [2] Escreva a função

```
void tPrint( TNode *r, int maxOccur );
```

que apresenta, em *standard output* por ordem alfabética crescente, uma seleção de palavras e as respetivas ocorrências. As palavras apresentadas são as que tiverem o número de ocorrências inferior ou igual a `maxOccur`. Se este parâmetro for 0, apresenta todas as palavras existentes.

5. [3 valores] Considere o programa seguinte.

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

#define MASK( n ) ( ~( ~0 << (n) ) )
#define SIZE_BIT( v ) ( sizeof( v ) * CHAR_BIT )

int match( int size, int val1, int val2 ){
    return ( val1 & MASK( size ) ) == ( val2 & MASK( size ) );
}

int shift( int value, int pos ){
    return value >> pos;
}

int find( int pattern, int size, int value ){
    for( int i = 0; i <  SIZE_BIT( value ) - size; ++i ){
        if( match( size, pattern, shift( value, i ) ) ){
            return i;
        }
    }
    return -1;
}

int main(int argc, char **argv ){
    if( argc != 3 ){
        exit( EXIT_FAILURE );
    }
    int p = atoi( argv[1] ), s = atoi( argv[2] );
    int x;
    while( scanf( "%d", &x ) == 1 ){
        int r = find( p, s, x );
        if( r == -1 ){
            printf( "Pattern not found\n" );
        }
        else{
            printf( "Pattern found at position %d\n", r );
        }
    }
    return EXIT_SUCCESS;
}
```

Admita que reorganiza o programa separando as funções nos três módulos seguintes: `match.c`, com a função `match`; `find.c`, com as funções `find` e `shift`; `pattern.c`, com a função `main`.

Considere os *header files*, `match.h`, e `find.h`, com as definições de tipo e assinaturas das funções relativas aos módulos `match.c` e `find.c`, respetivamente.

a) [0,5] Identifique as funções, se houver, em que seja adequado usar o atributo `static`.

b) [0,5] Indique em que ficheiro deve ser colocada cada uma das definições de macro, `MASK` e `SIZE_BIT`.

c) [1] Tendo em conta a sua resposta à alínea a), apresente a lista produzida pela ferramenta `nm` para cada um dos três módulos compilados, `match.o`, `find.o` e `pattern.o`, indicando os nomes dos símbolos e a respetiva classificação, `t` (*private code*), `T` (*public code*) ou `U` (*undefined*). Indique, por cada símbolo indefinido, qual é o módulo que o resolve ou, em alternativa, se é resolvido pela biblioteca normalizada.

d) [0,5] Indique os módulos fonte que devem incluir cada um dos *header files*; escreva o *header file* `find.h` com as definições adequadas e o controlo de inclusão múltipla.

e) [0,5] Considerando os módulos e os *header files* referidos, escreva o *makefile* para gerar o executável com o nome `pattern`.